

FIG. 1

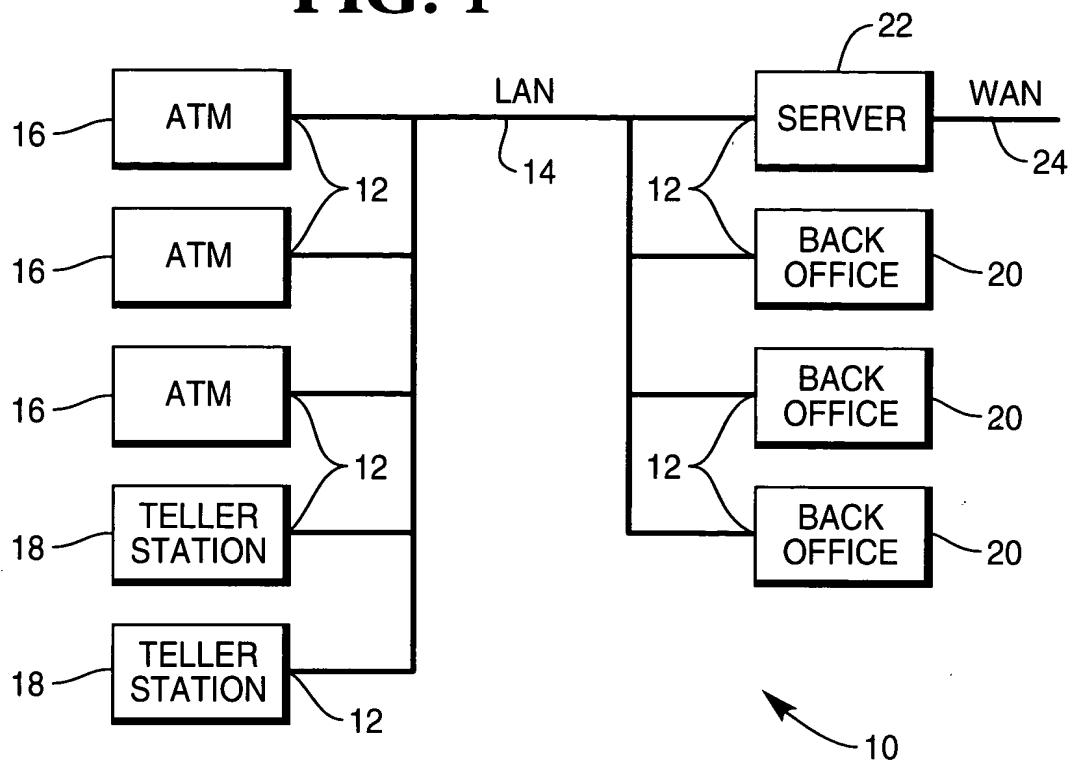


FIG. 2

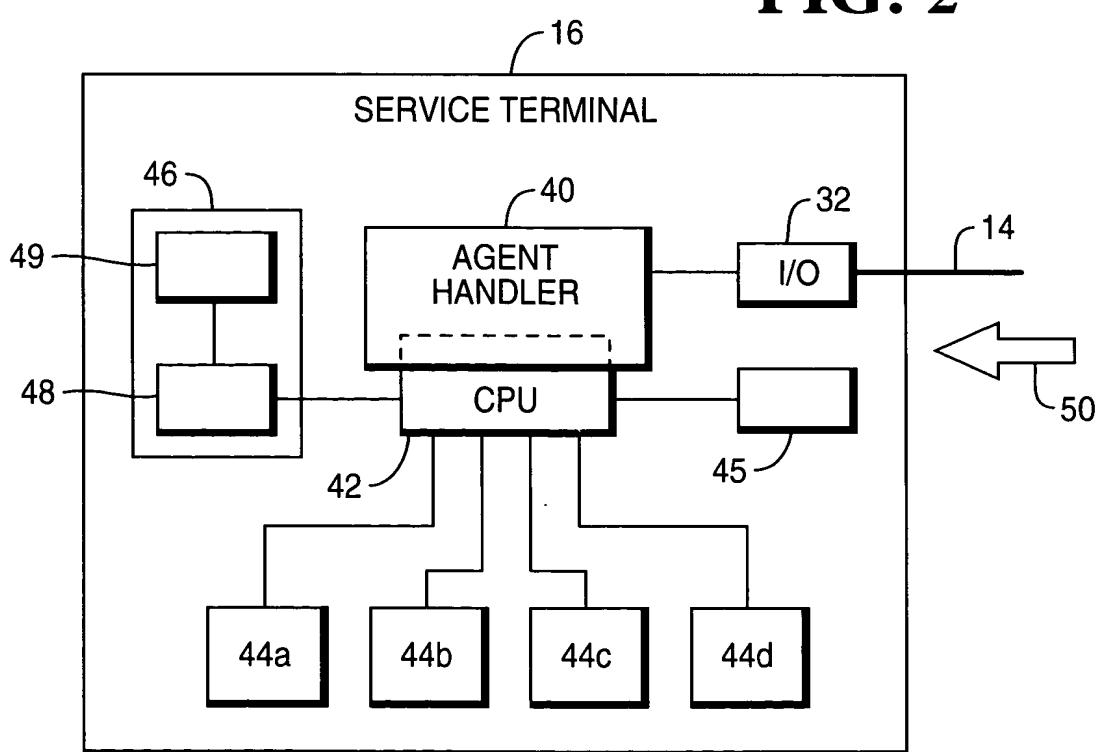


FIG. 3

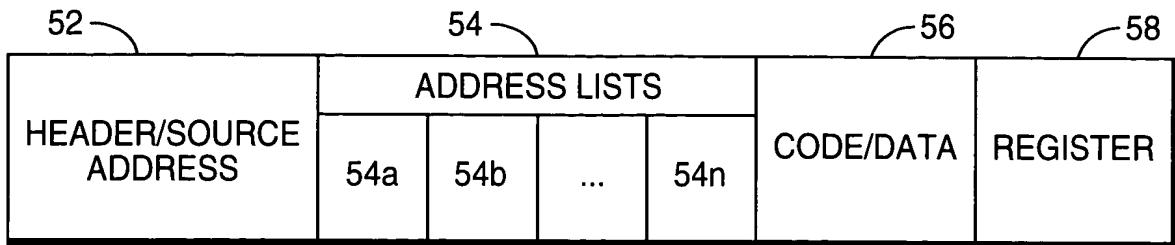


FIG. 4

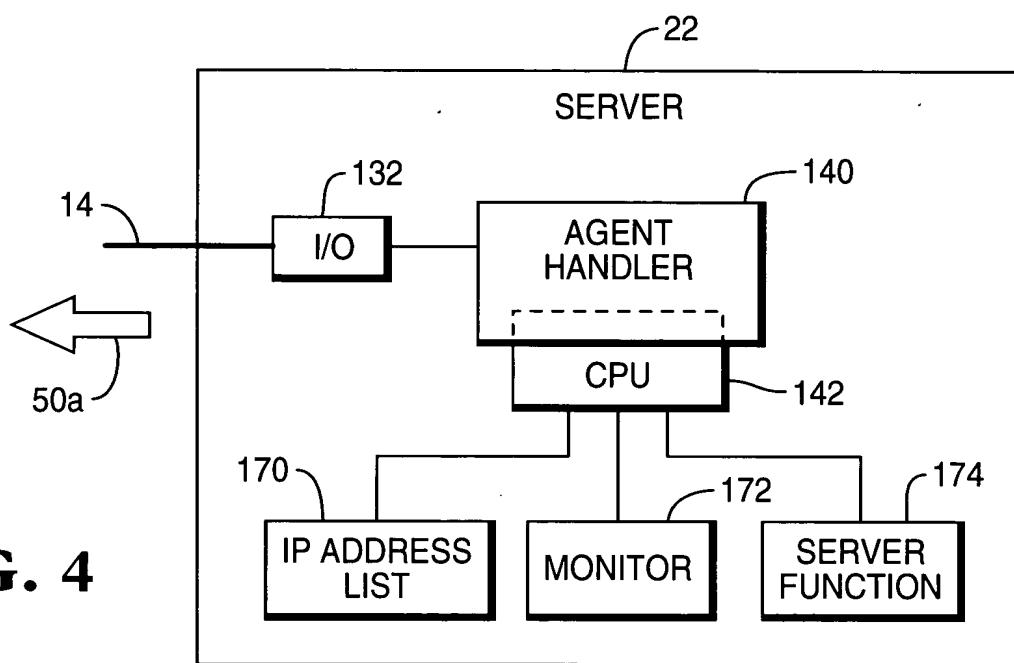


FIG. 5

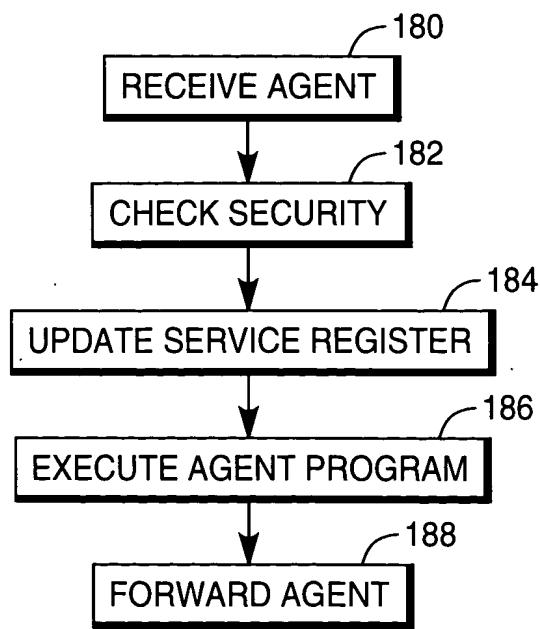


FIG. 6a

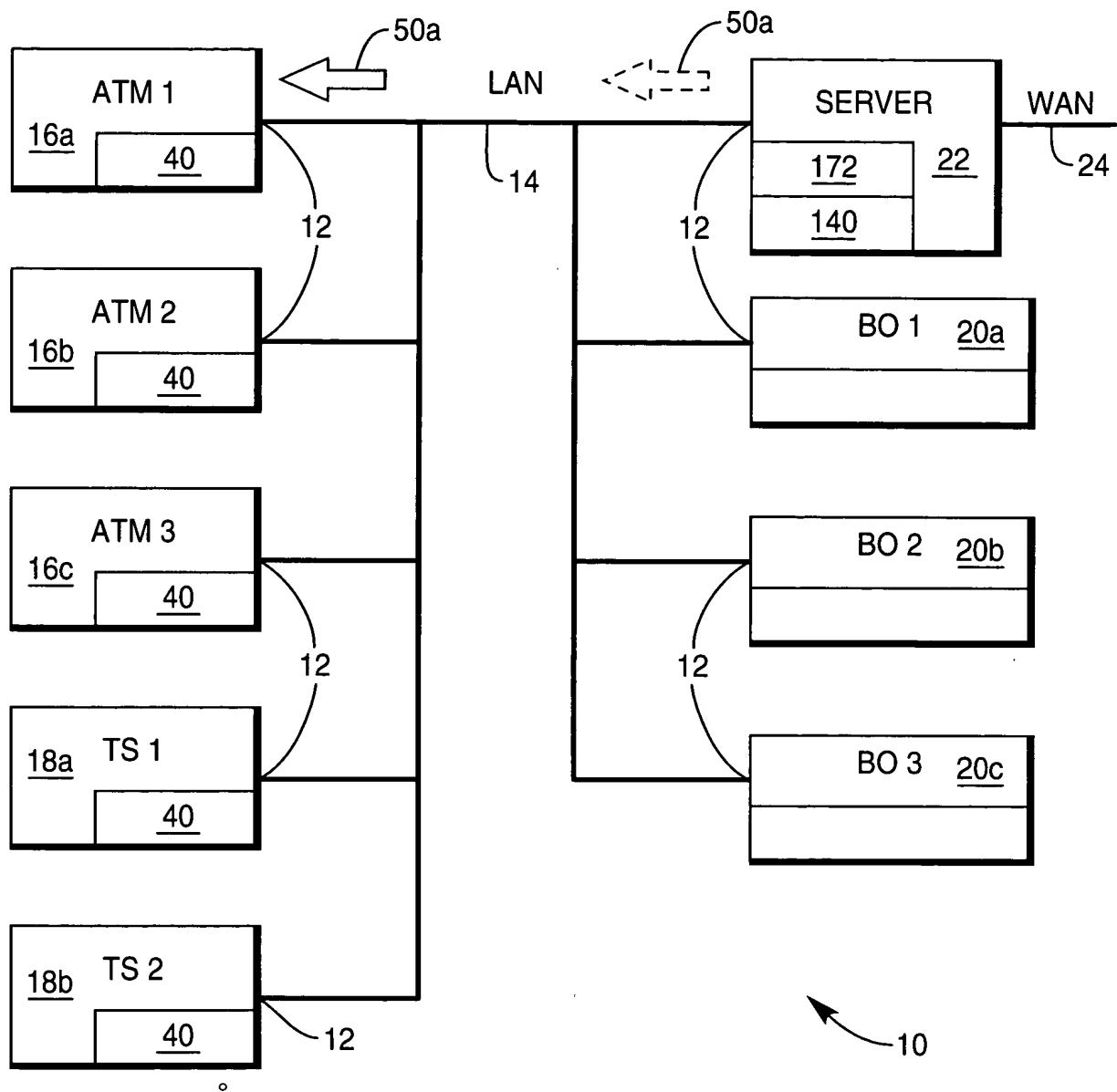


FIG. 6b

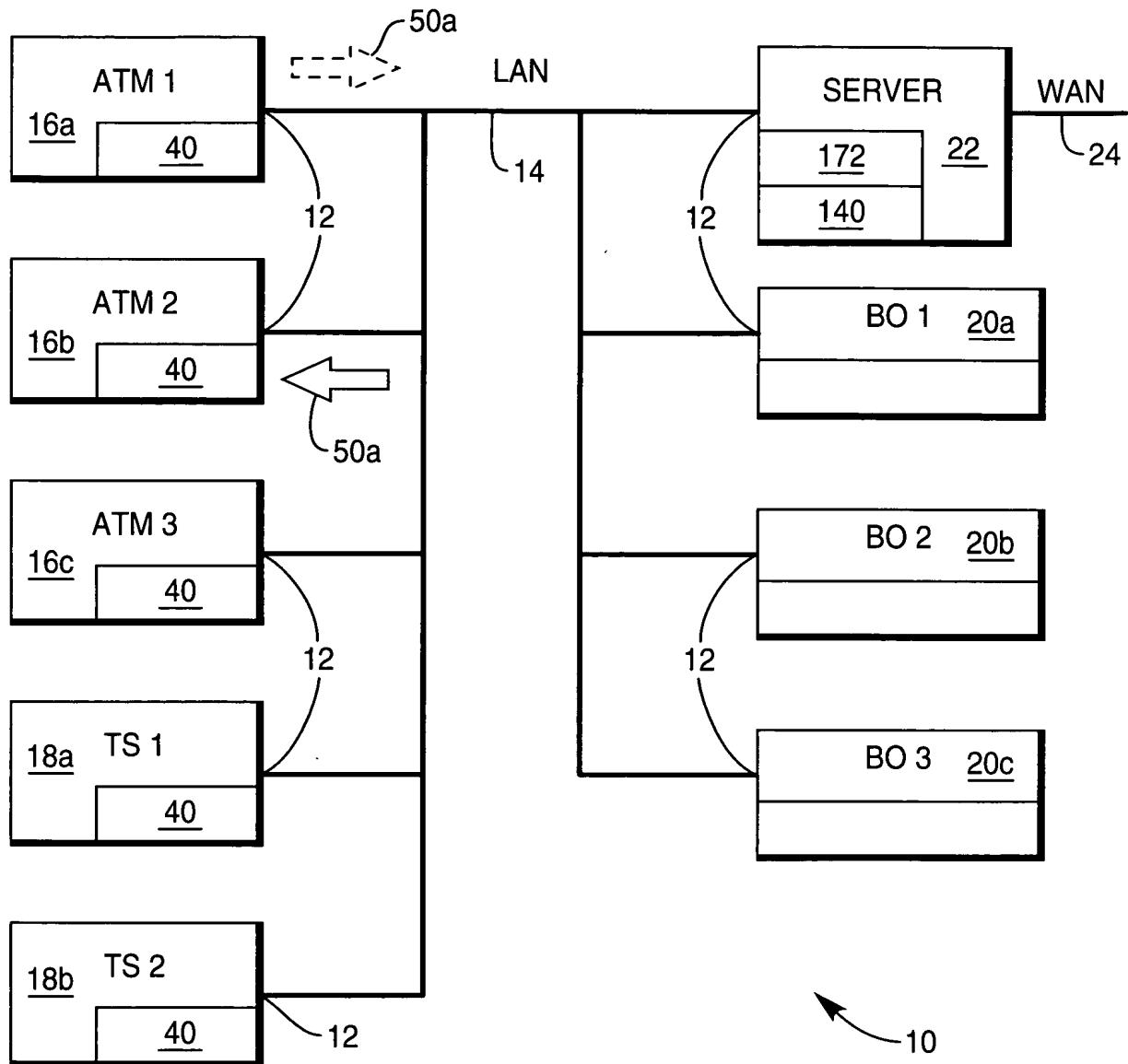


FIG. 7

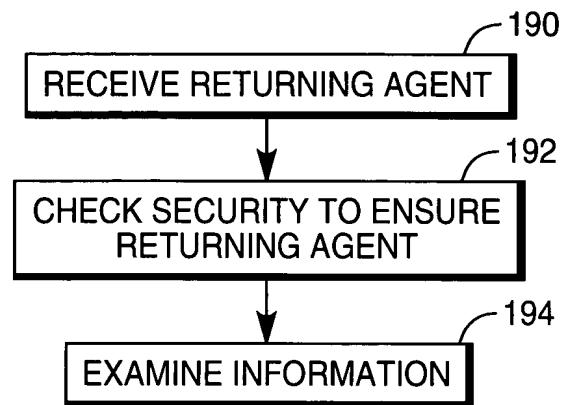


FIG. 9

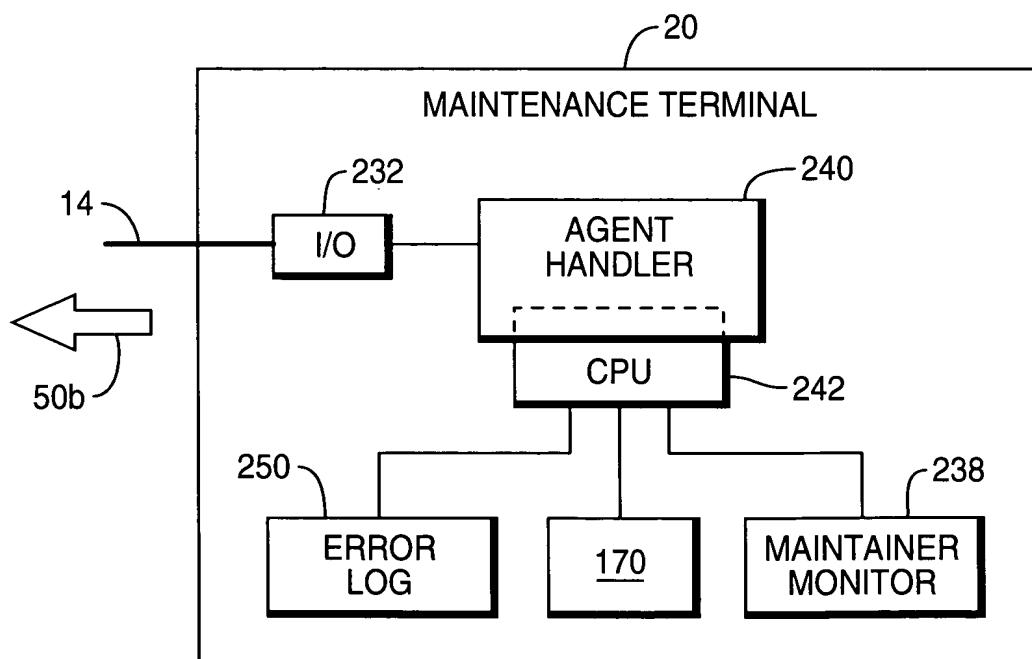


FIG. 6c

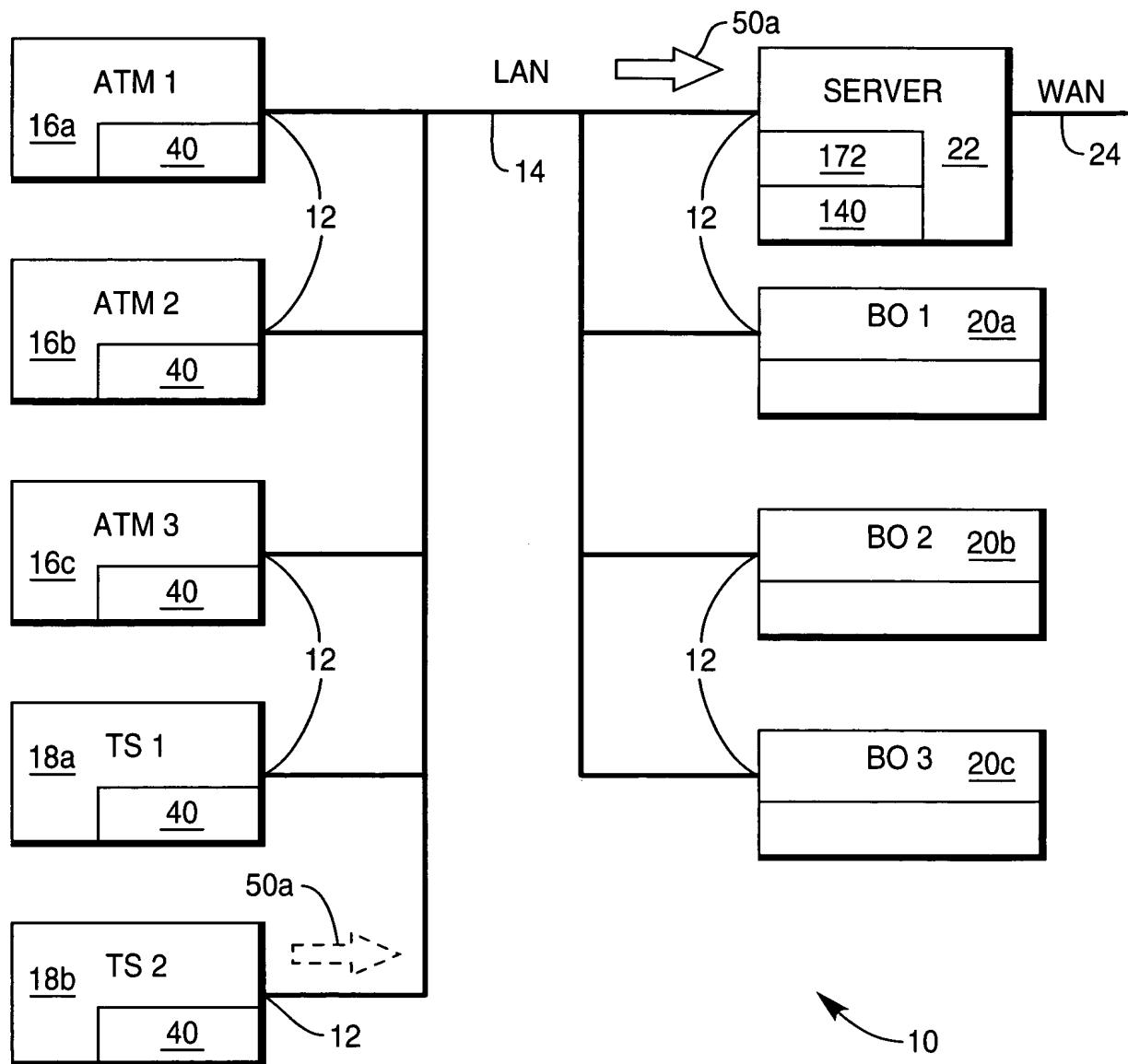


FIG. 8a

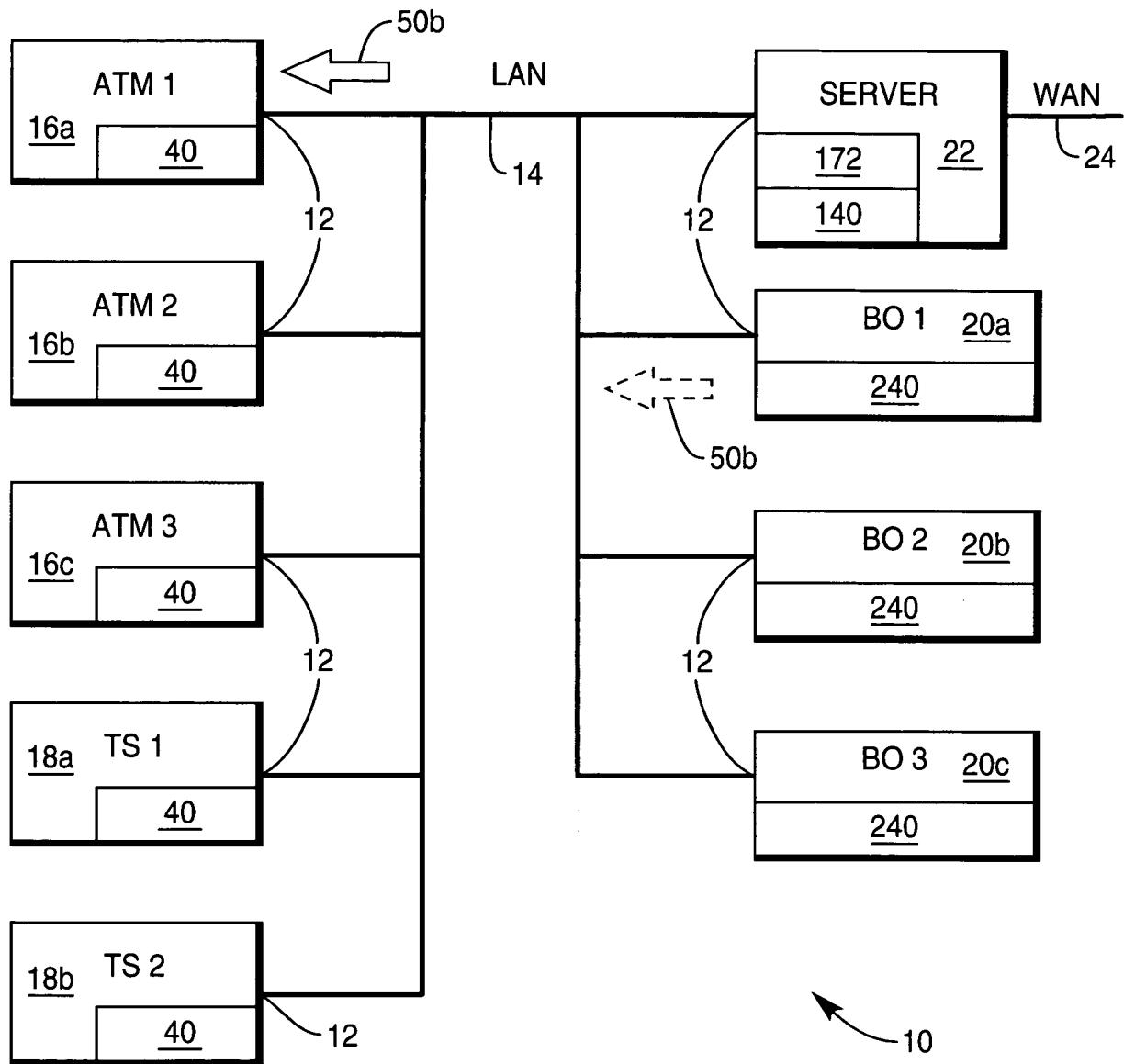


FIG. 8b

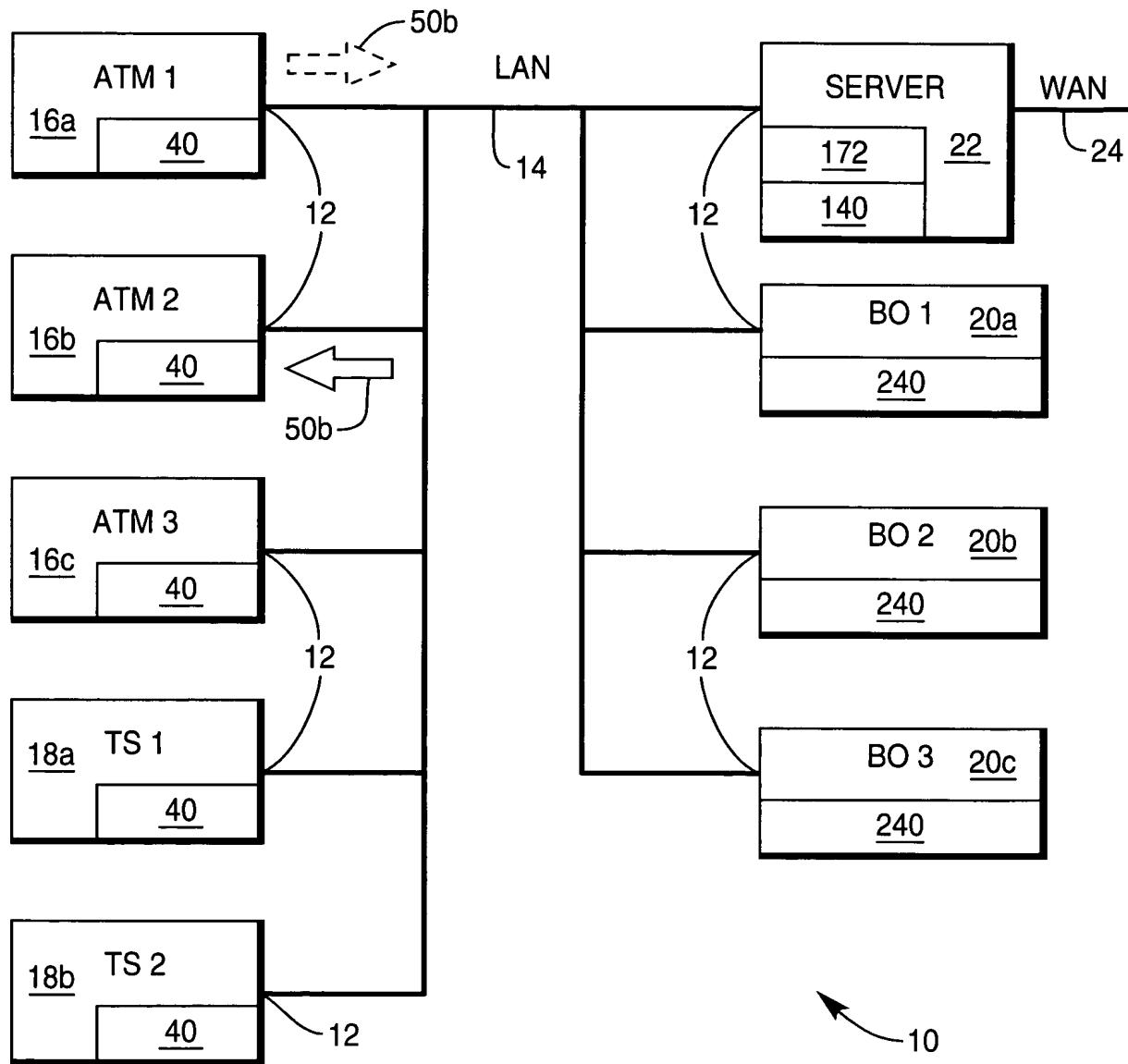


FIG. 8c

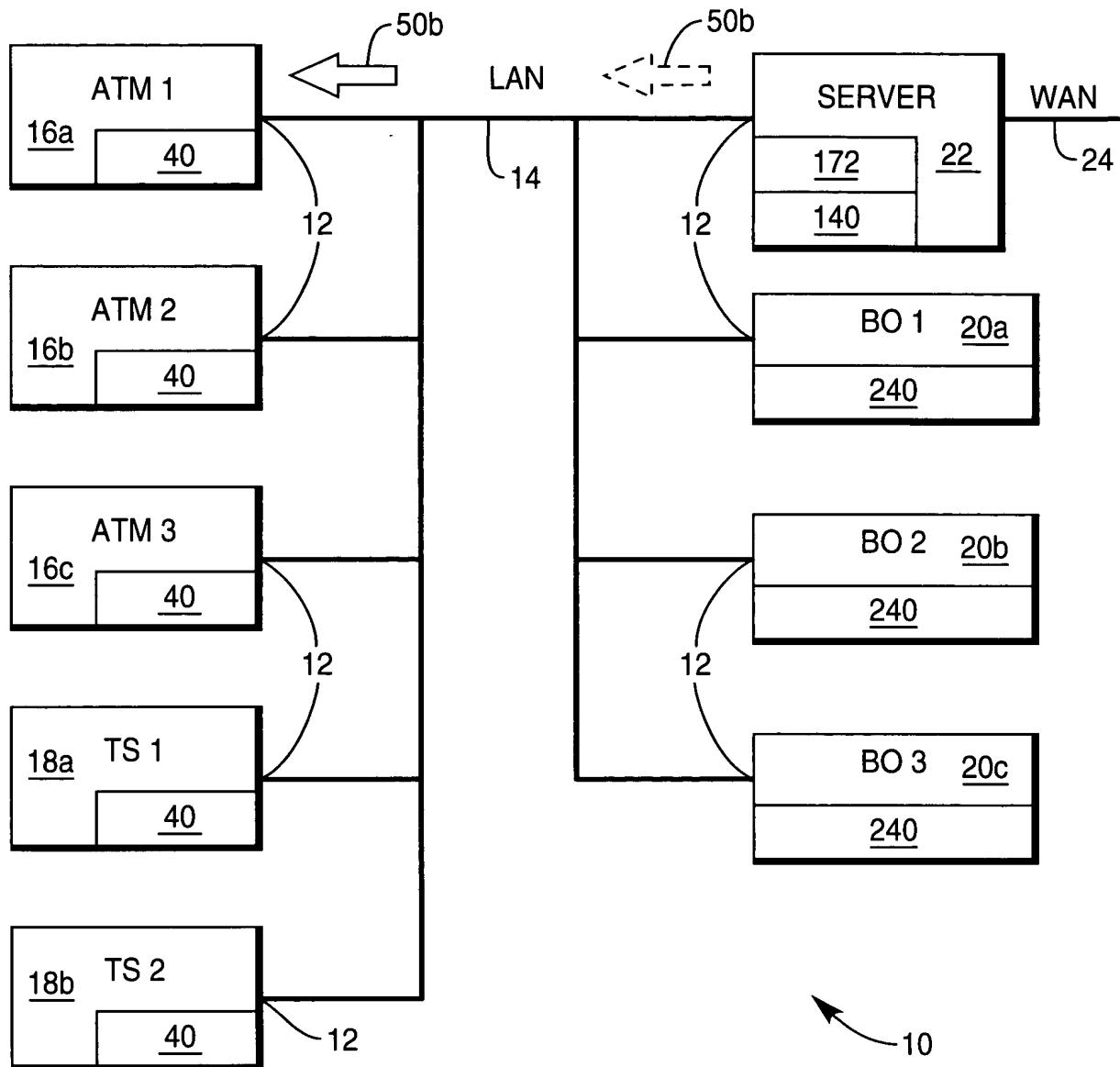


FIG. 10

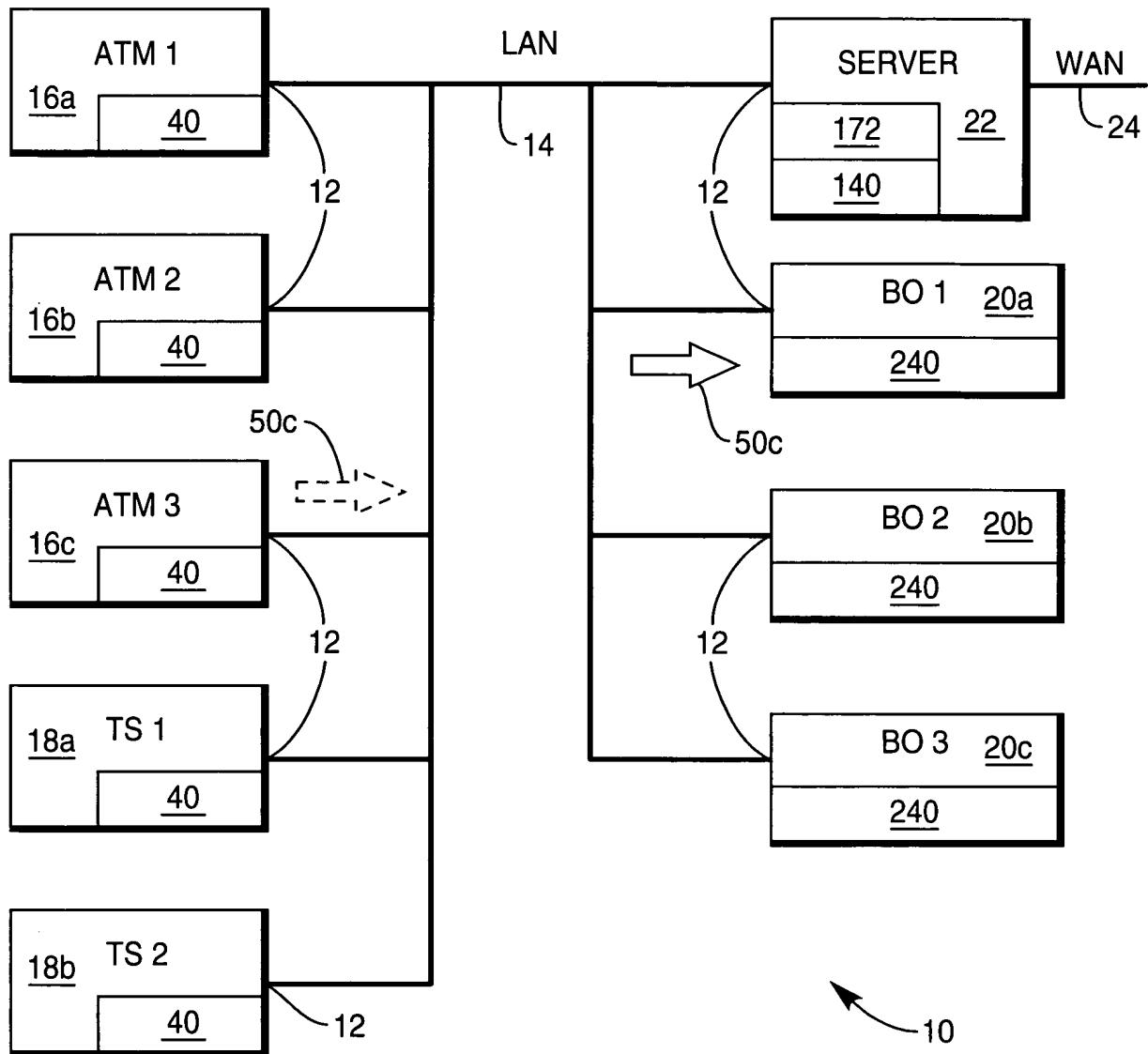


FIG. 11a

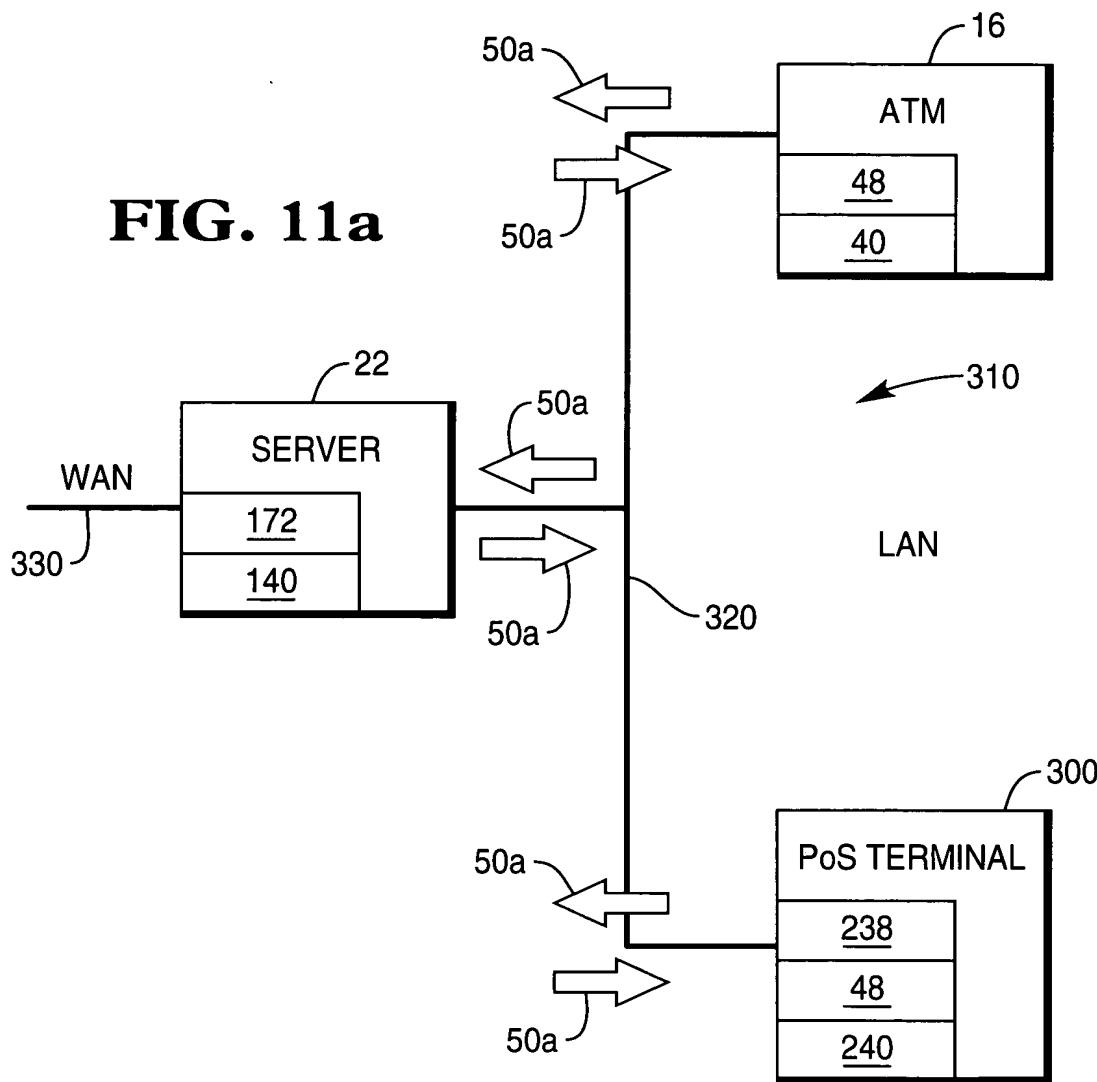


FIG. 11b

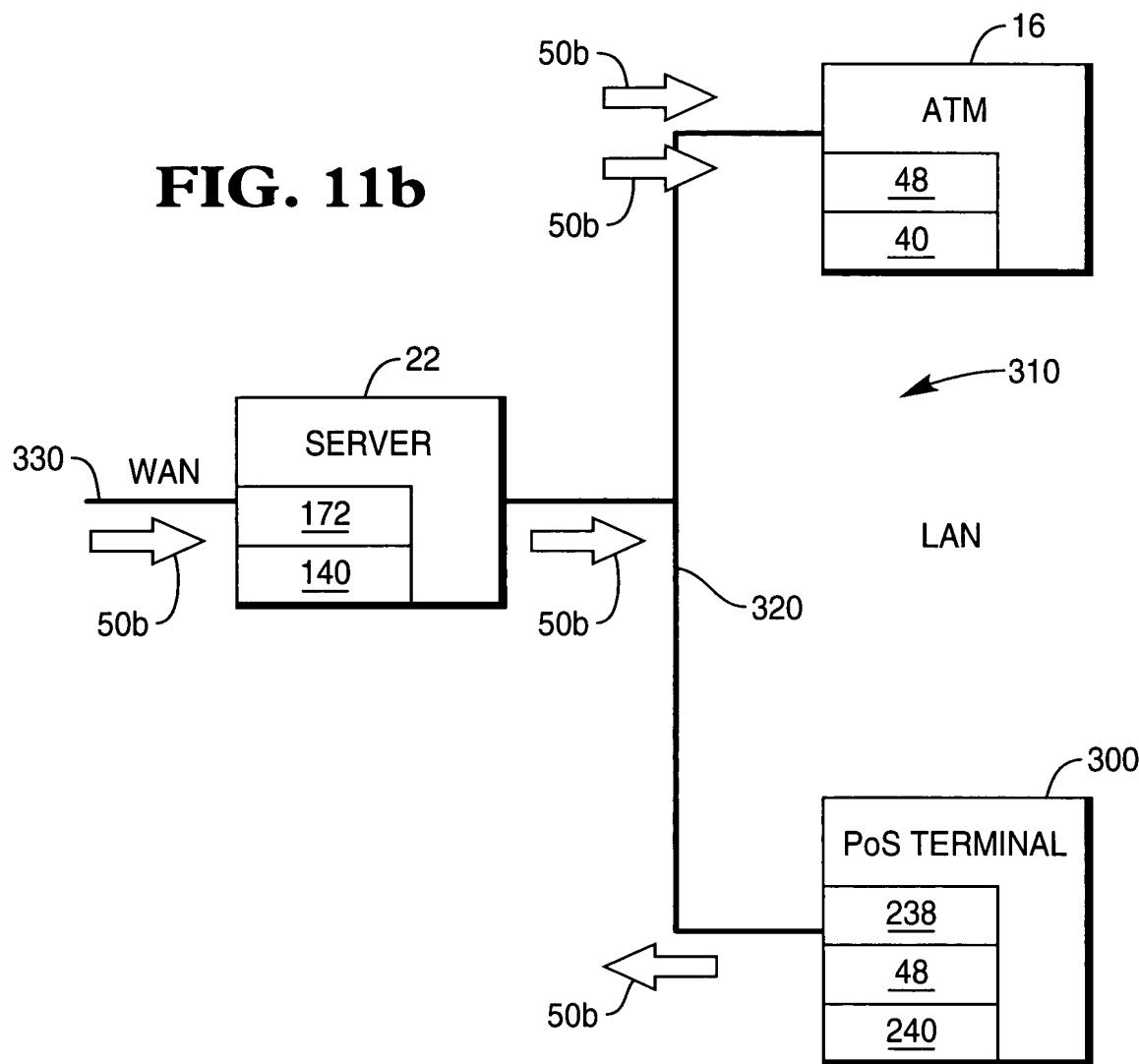
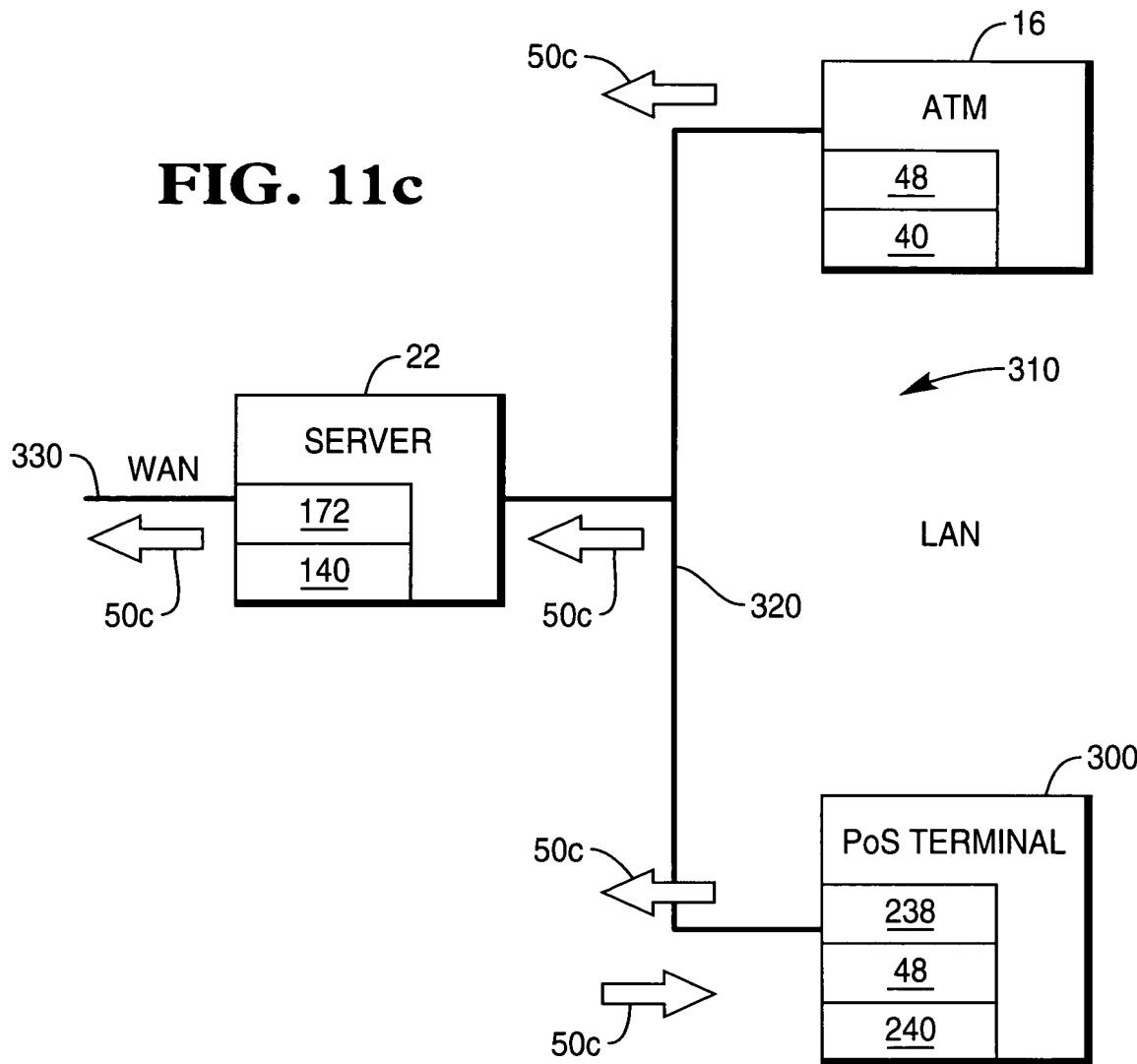


FIG. 11c



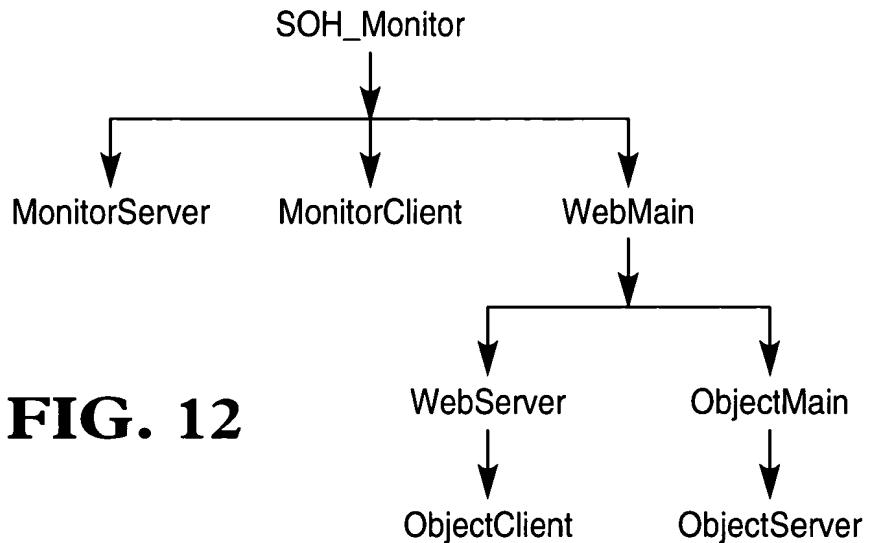


FIG. 12

```

public class ModuleDetails implements Serializable
{
    /** Modules port number */
    int PortNumber;

    /** Modules name */
    String ModuleName;

    /** Modules IP address */
    String IPAddress;

    /** Modules State of Health */
    String SOH;
}
  
```

FIG. 13

FIG. 14

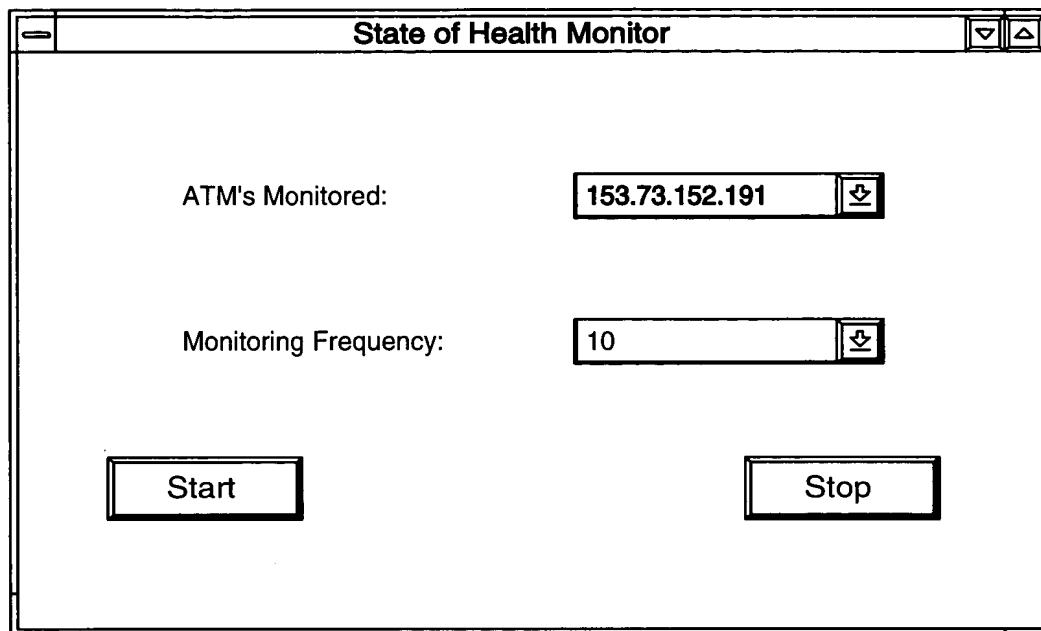


FIG. 15

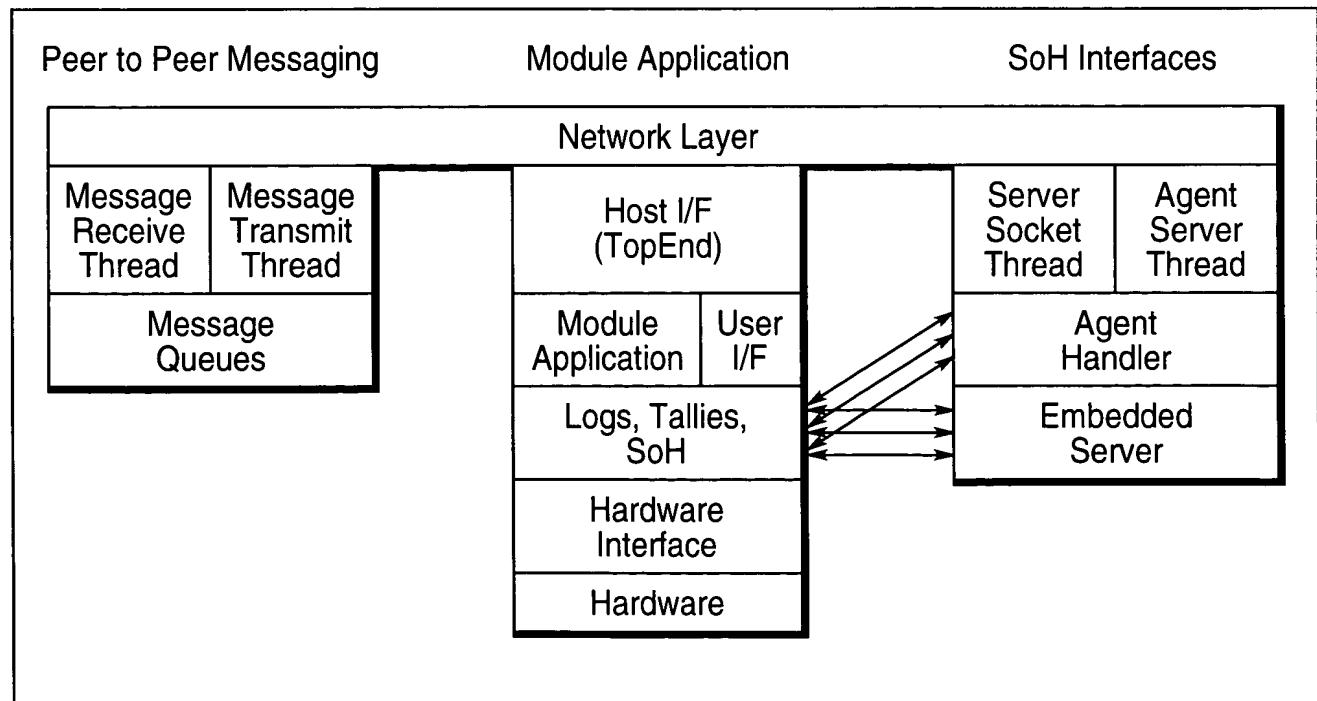


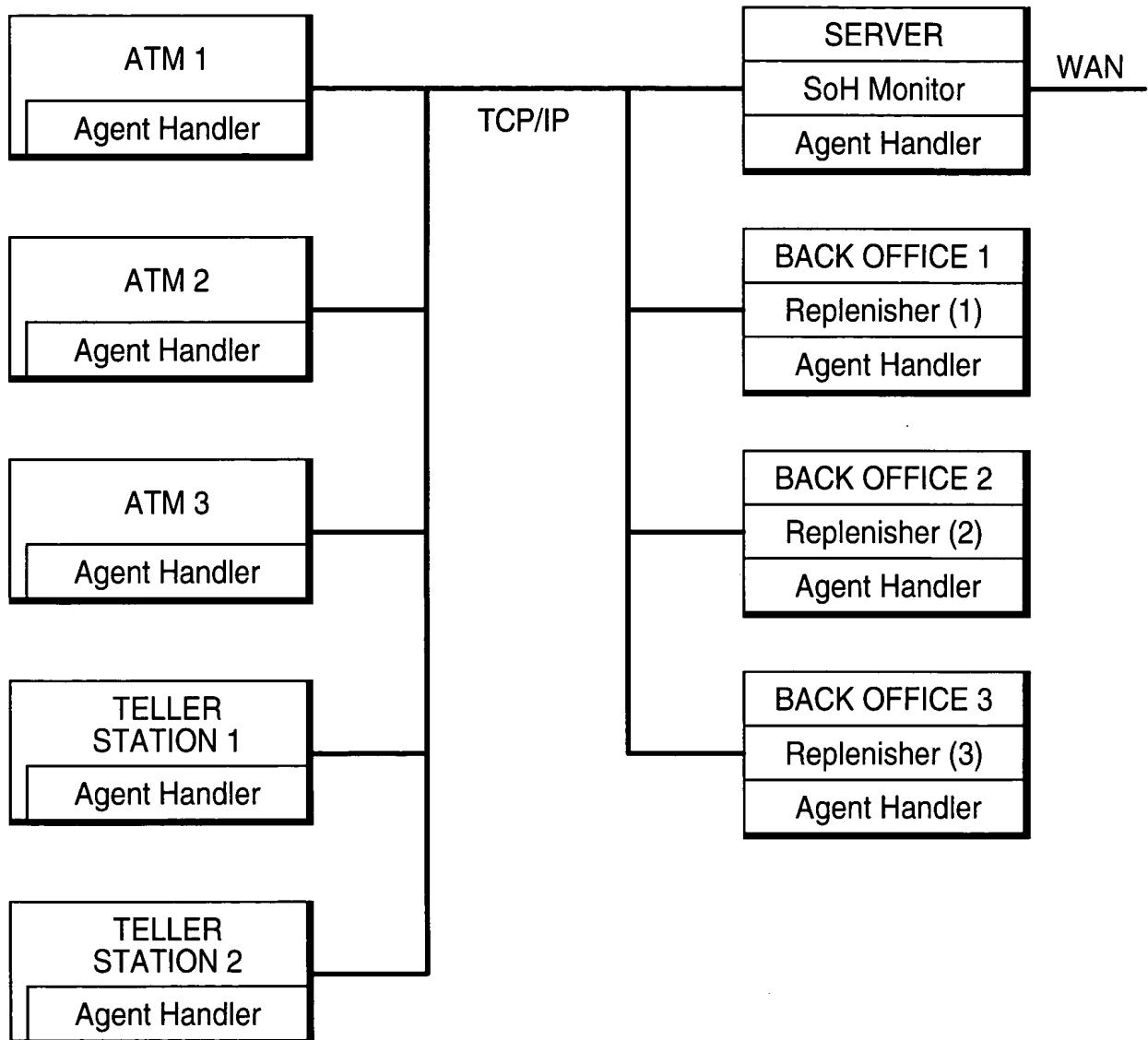
FIG. 16

```
public void Searching(String temp)
{
    if (temp.equalsIgnoreCase("SHTR JAM"))
    {
        Log[0] = Log[0] +1; // Incrementing the log if a match is found
        if (Log[0] >= 0) // Threshold = 0
            error.setErrorArray(0,"Shutter Jam"); // If the threshold is broken
            then
                //set array location0 to
                'Shutter Jam'
    }

    // Calls the function that will send the Error Array to the Regional
    // Server and generate and Alert Agent if the error is fatal
    send();
}
```

FIG. 17

BRANCH OFFICE NETWORK



STATE OF HEALTH MONITORING

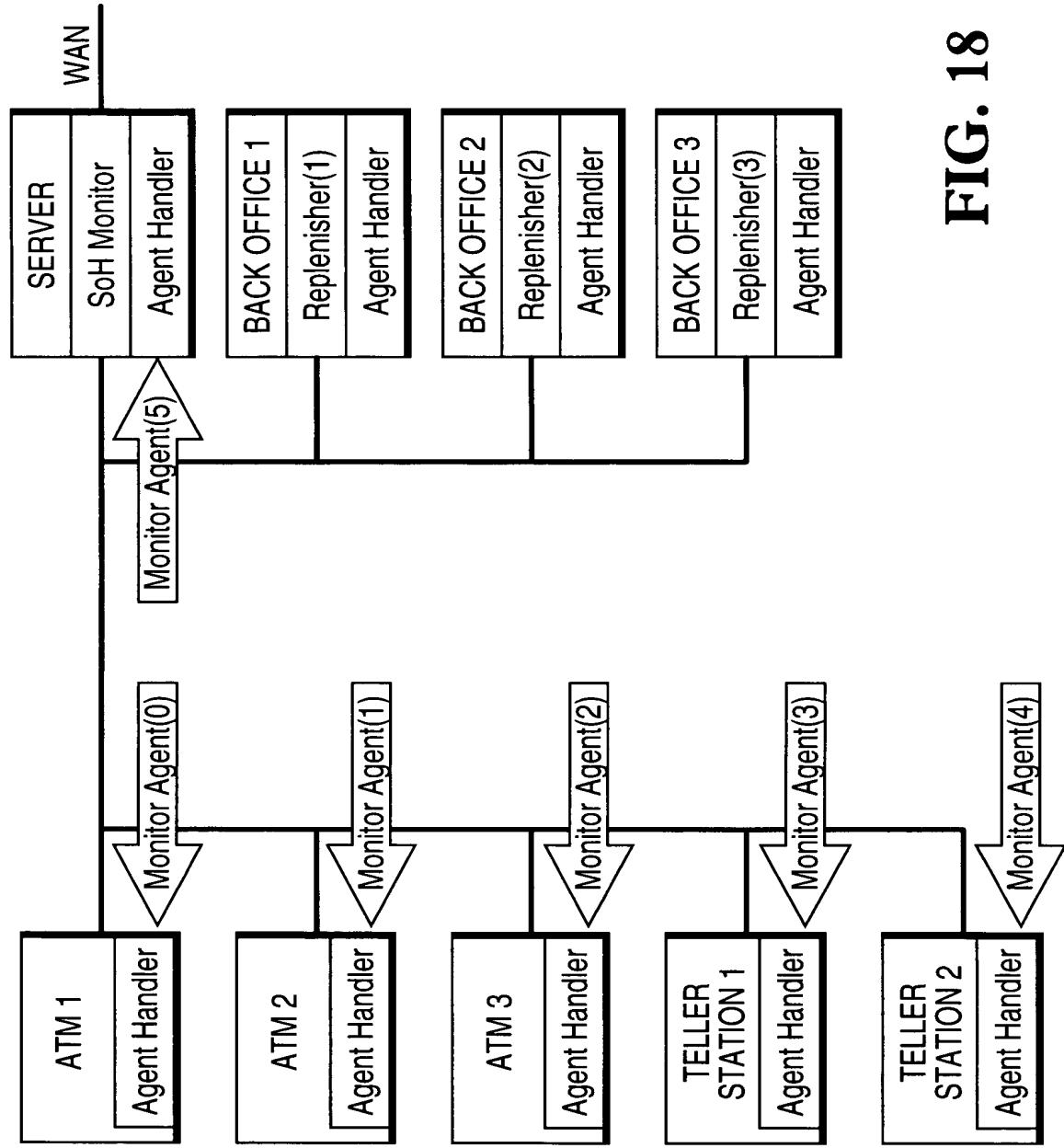


FIG. 18

FIG. 19

```
/** Determines whether or not to send out a Alert Agent */
public void processReturningAgent()
{
    int count =0;
    Vector ATMsToVisit = new Vector(1,1);
    AlertAgent aAgent;

    ATMsToVisit = mAgent.getATM();
    int size = ATMsToVisit.size();

    while (count < size) // Extracts the records one at a time and examines the State of Health
    {
        ModuleDetails Mtemp = (ModuleDetails)ATMsToVisit.elementAt(count);
        String tempSOH = Mtemp.getSOH(); // Gets the modules state of Health

        if (tempSOH.equals("Healthy"))
        {
        }
        else
            // Create an alert agent and initialise it with the Replenisher Location Details.
            {
                agent = new AlertAgent(RepIP, RepPort,Mtemp);

                // Get the Internet address and the port number
                // of the first Replenisher String nextAddress = RepIP[0];
                int nextPort = RepPort[0];

                // Send out the Alert Agent
                MonitorClient client = new MonitorClient(nextAddress,nextPort,aAgent);
            }
        count++;
    }
}
```

FIG. 20

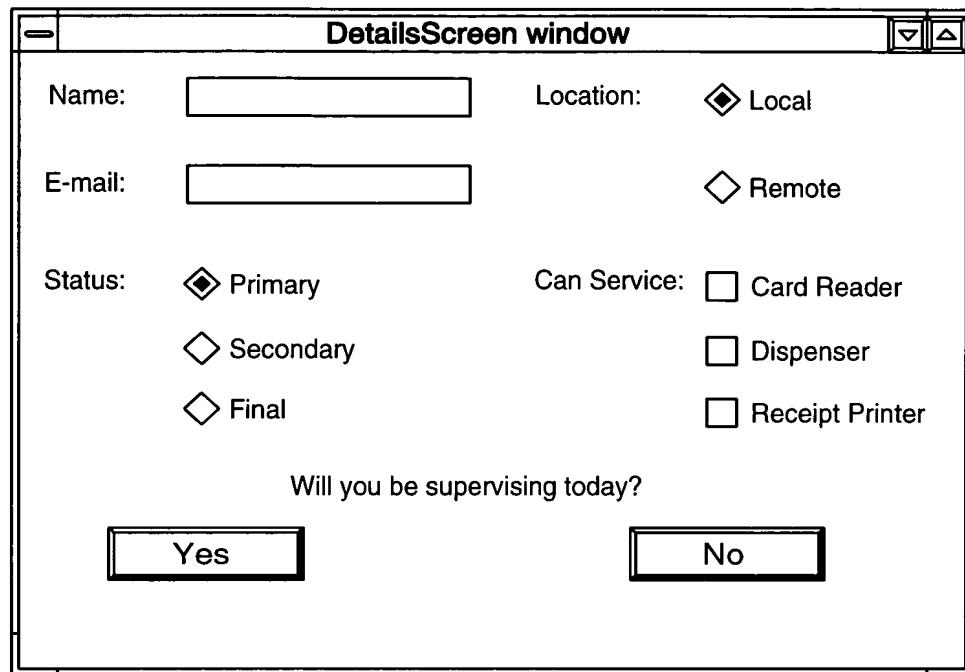


FIG. 22

```
public void StoreData()
{
    int count =0;
    boolean flag = false;
    String temp = sAgent.getReplenishersIP();

    while(RepIP[count] != null) // Finds the next free space to add the Replenisher data to
    {
        count++;
    }

    RepPort[count] = sAgent.getReplenishersPort(); // Adds the new data to the arrays
    RepIP[count] = sAgent.getReplenishersIP();
    sAgent.SendOutServiceAgent(); // Passes the service agent onto the next module
}
```

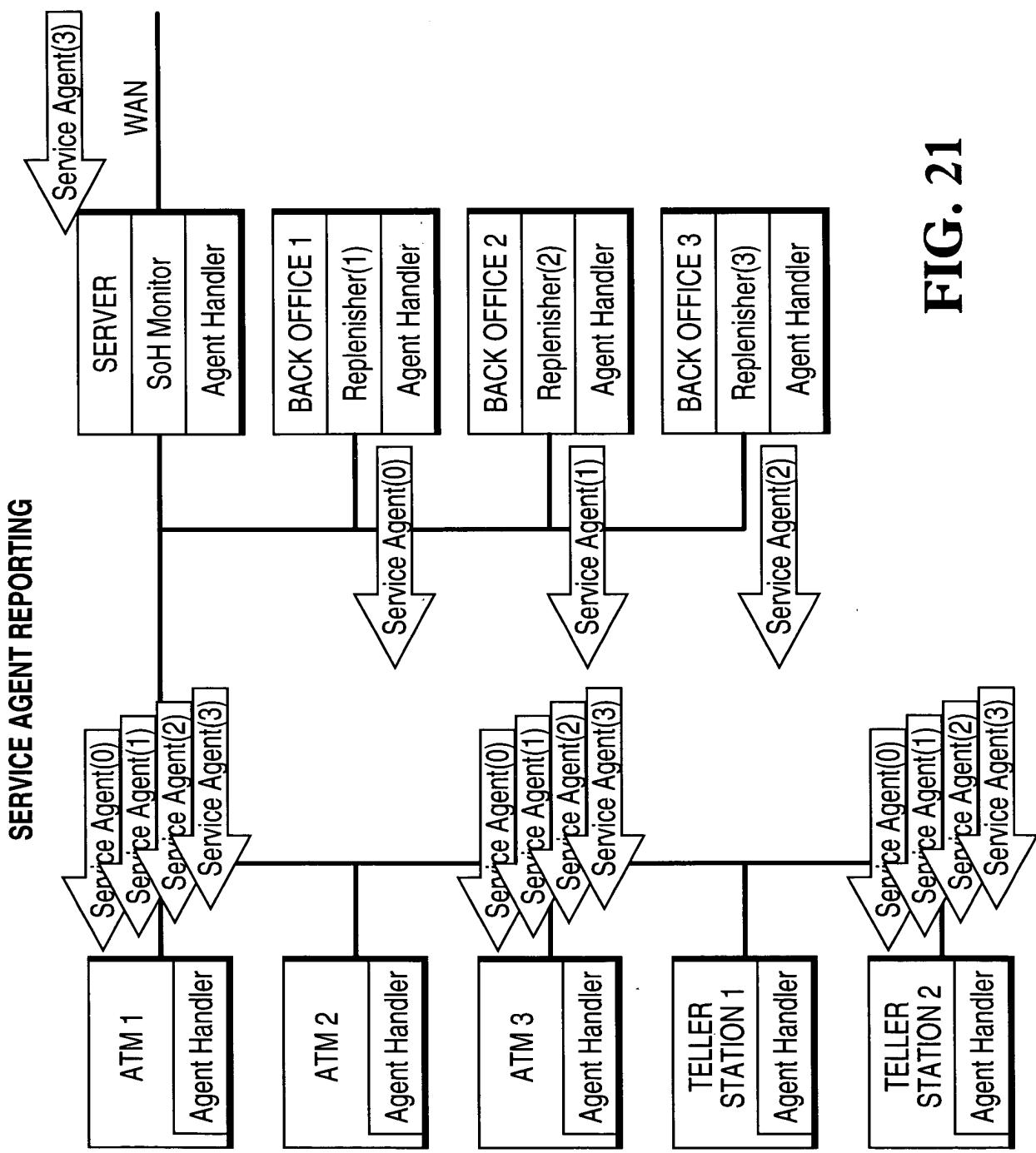


FIG. 23

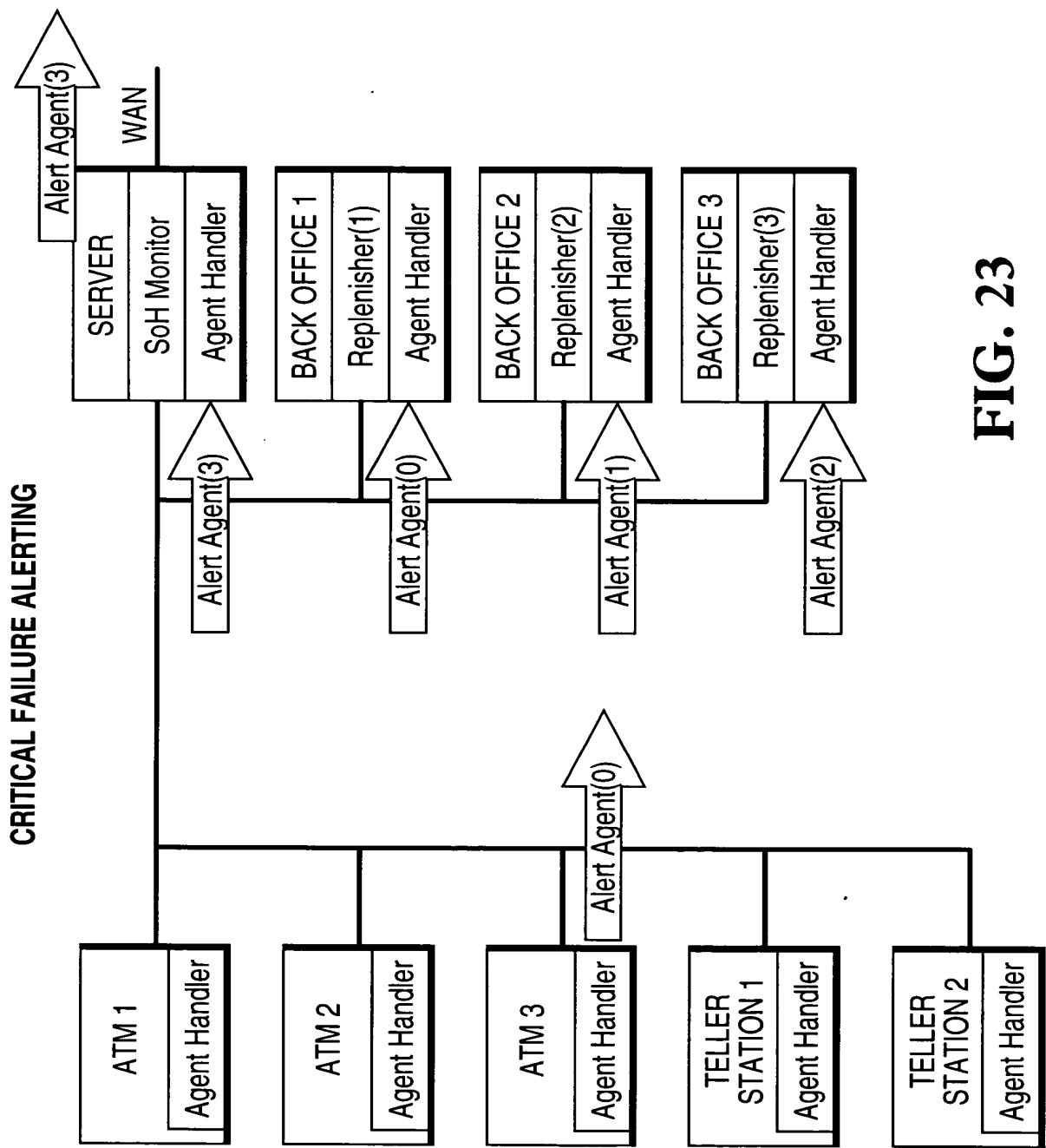


FIG. 24a

```
public void processServicersInfo(RepDetails rep, String IP, int port)
{
    DisplayError dError;
    AlertAgent agent;

    RepDetails details = new RepDetails();
    details = rep; // A local copy of the class RepDetails, this stores all the Replenishers
details
    int count =0;
    int counter =0;
    CurrentIP = IP;
    CurrentPort = port;

    String tempLocation = details.getLocation(); // Gets the Replenishers location
    String tempStatus = details.getStatus(); // Gets the Replenishers status
    String[] tempService = details.getServices(); // Gets the list of what the Replenisher can
fix

    if (tempLocation.equalsIgnoreCase(LookForLocation))
    {
        if (tempStatus.equalsIgnoreCase(LookForStatus))
        {
            if (tempService[0].equalsIgnoreCase("Dispenser"))
            {
                dError = new DisplayError(ErrorIP, ErrorModule, ErrorSOH,this);
            }
            else if (tempService[1].equalsIgnoreCase("Card Reader"))
            {
                dError = new DisplayError(ErrorIP, ErrorModule, ErrorSOH,this);
            }
            else if (tempService[2].equalsIgnoreCase("Printer"))
            {
                dError = new DisplayError(ErrorIP, ErrorModule, ErrorSOH,this);
            }
        }
        if (tempStatus.equalsIgnoreCase("Primary"))
            LookForStatus = "Secondary";
        if (tempStatus.equalsIgnoreCase("Secondary"))
            LookForStatus = "Final";
    }
}
```

FIG. 24b

```
if (tempStatus.equalsIgnoreCase("Final"))
{
    LookForStatus = "Primary";
    if (tempLocation.equalsIgnoreCase("Local"))
    {
        LookForLocation = "Remote";
    }
}
NumberOfServicers++;
}
else
{
    int size = 0;
    while (IPList[size] != null) // Gets the number of Replenishers in the list
    {
        size++;
    }

    if (NumberOfServicers == size) // When all the Replenishers have been visited, go
back
    {
        int nextport = PortList[0]; // Set to the starting port number
        String nextIP = IPList[0]; // Set to the starting IP number
        String signal = "Alert Agent";
        Client client = new Client(nextIP, nextport, signal, this); // Send out the agent
    }
    else
    {
        while (IPList[counter] != CurrentIP) // Find the array position of the Replenisher
        {
            counter++;
        }

        if (IPList[counter].equalsIgnoreCase(CurrentIP))
        {
            int nextport = PortList[counter++]; // Move on to the next Replenisher if
            String nextIP = IPList[counter++]; // nothing suitable here
            String signal = "Alert Agent";
            Client client = new Client(nextIP, nextport, signal, this);
        }
    }
    NumberOfServicers++;
}
```

FIG. 25

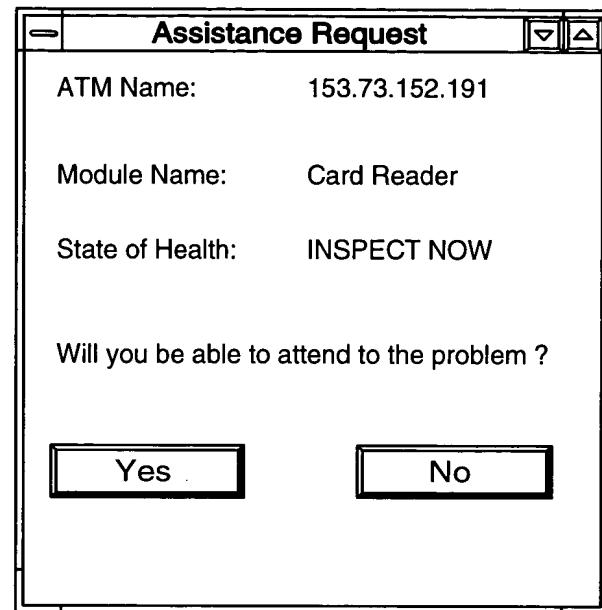
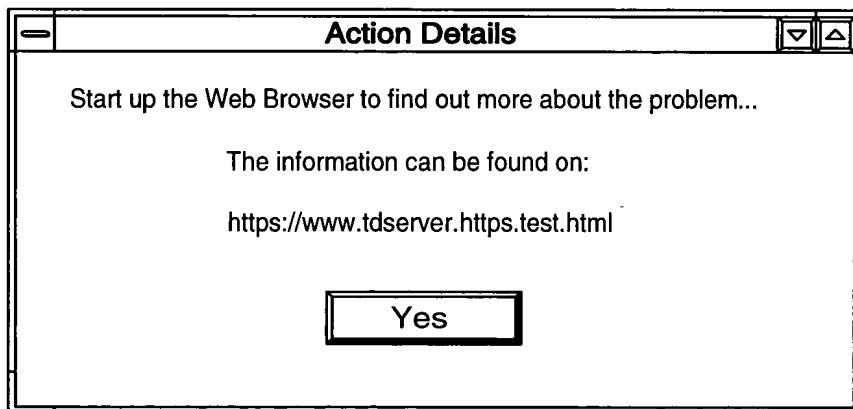


FIG. 26



ATM MODULE

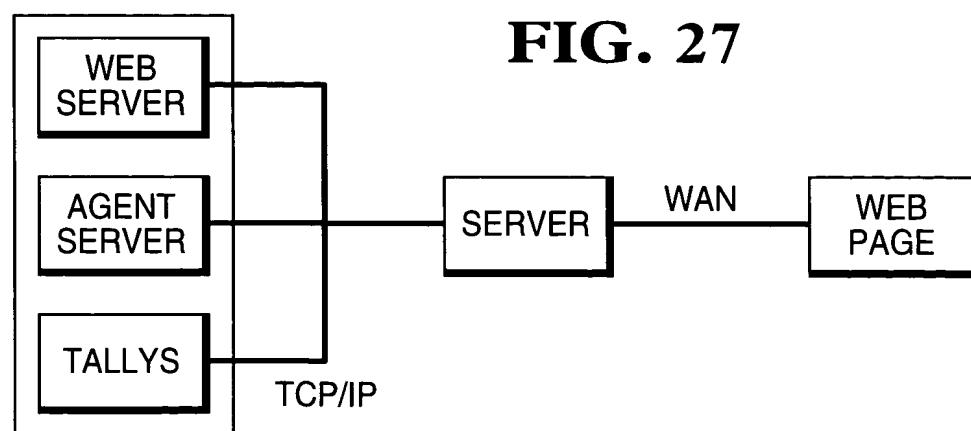


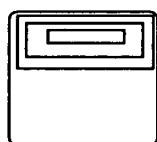
FIG. 27

FIG. 28

```
If (temp.equalsIgnoreCase("Shutter Jam"))
{
    Description[count] = "The shutter has jammed " + "open or closed. This " +
                           " was detected when a card was" + "being accepted or ejected";

    Check[count] = "Shutter, Card Reader";
    StateOfHealth = "INSPECT NOW";
}
```

FIG. 29



Available

FIG. 30



192.23.234.195



FIG. 31

```
public void runServer()
{
    int sPort = 6050; // initialises this value to 6050
    ServerSocket Serv;

    try
    {
        // Set the socket to be monitored
        Serv = new ServerSocket(sPort);

        // if the thread does not link to anything then create one and start it running
        if (runner == null)
        {
            // run the ServerThread as a new thread
            runner = new AppletServer(Serv,this);

            // start the thread runner running
            runner.start();
        }
    }
    catch (Exception e)
    {
    }
}
```

FIG. 32

```
public void run()
{
    Object temp = null;
    String tempString = null;

    while (true)
    {
        try
        {
            // waits on the server until a message is received
            Socket server = Serv.accept();

            // creates a new stream to connect to and bind
            ObjectInputStream in = new ObjectInputStream(server.getInputStream());

            // reads in the object and changes it too a String
            temp = in.readObject();
            tempString = temp.toString();

            if (tempString.equalsIgnoreCase("Service Agent"))
            {
                SAgentPresent = true;

                // Reads in the service agent
                sAgent= (ServiceAgent)in.readObject();
            }
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

FIG. 33

FIG. 34

```
public void SendAgent()
{
    // Set Output To Send Agent To First Server
    try
    {
        // try and connect to Socket destName.destPort
        Socket s = new Socket(destName, destPort);

        // create an ObjectOutputStream (out)
        ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());

        // Got a connection Send Agent To Server
        // Sends out the corresponding agent
        if (state.equalsIgnoreCase("Service Agent"))
        {
            out.writeObject(state);
            out.writeObject(sAgent);
        }

        // Close outgoing port
        s.close();
        out.close();
    }
    catch (Exception e)
    {
    }
}
```

FIG. 35a

Functional Block	Class Name	Description
SoH	AppletCenter.java	Handles all the displaying and passing of data to the Applet. It controls what images etc. are displayed and when they are displayed.
	AppletMain.java	Used to check that the thread is not already running and that the server can start running. Also used to pass data between the AppletCenter class and the AppletServer class.
	AppletServer.java	Listens on a specified server connection and waits for incoming messages. It listens for messages from the Regional Server. Once a message has arrived it takes the corresponding actions.
	ControlPanel.java	Holds navigation buttons and a Break button that is used only for demonstration purposes.
	Errors_Fixed.java	Displays the Fix Errors screen and all the details about an error. Any of the errors can be fixed by selecting one and pressing the OK button. This causes the ATM to reset itself and the error to be removed from the error records.
	FixMoney.java	Help screen on how to fill the ATM with money.
	Ink.java	Help screen on how to replace ribbon.
	Paper.java	Help screen on how to fill the ATM with paper.
	Knife.java	Help screen on how to replace the knife.
	PurgeBin.java	Help screen on how to empty the purge bin.
	Record.java	Stores all the details about one error. An array of these is used to pass error data between the various classes and store all current error information.
	Screen0.java	Displays the components of the main screen on the Applet.
	Screen1.java	Displays the components of the Replenishers screen on the Applet.
	Screen2.java	Displays the components of the Field Engineers screen on the Applet.

FIG. 35b

Functional Block	Class Name	Description
Regional	Agent.java	Class that is inherited by all the various agents.
	AlertAgent.java	Agent that is produced when an error occurs and takes this data to the Replenisher to ask for help.
	ConfirmMess.java	Class that displays the whereabouts of the Applet.
	DisplayError.java	Class that displays the components of the error and ask the Replenisher for help.
	ModuleDetails.java	Stores the individual details of an ATM module. This information is vital in the movement of the agents around the network.
	MonitorAgent.java	Agent that is used to constantly monitor the State of Health of all the ATM modules. It has also the intelligent to move around the network.
	MonitorClient.java	This class has the ability to bind to ports on certain terminals and send out data for them. This class has the ability to send out all the different agents.
	MonitorServer.java	Listens on a specified server connection and waits for incoming messages. It listens for Service Agents, Returning Monitor Agents and ATM modules registering. Once a message has arrived it takes the corresponding actions.
	SOH_Monitor.java	Main class for this package. Used to handle all the incoming data and execute the calls that will send out the Monitor Agents.
	ObjectClient.java	This class has the ability to bind to ports on certain terminals and send out data for them. It is responsible for sending requests to an ATM module WebServer.

FIG. 35c

Functional Block	Class Name	Description
Regional - Cont'd	ObjectServer.java	Listens on a specified server connection and waits for incoming messages. Listens for messages and reset acknowledgements from the Embedded WebServers. Once a message has arrived it takes the corresponding actions.
	ObjectMain.java	Used to check that the thread is not already running and that the server can start running.
	ServiceAgent.java	Agent that will take the Replenishers Internet address and agent handler port number to the Regional Server and all of the registered ATM modules.
	StringProcessor.java	Adds additional information to be sent out along with the error data to the Applet.
	WaitAWhile.java	This is a thread that is used to sleep if an ATM has not registered and a Replenisher has logged in.
	WebClient.java	This class has the ability to bind to ports on certain terminals and send out data for them. Can send out serial data streams to the AppletServer.
	WebMain.java	Used to check that the thread is not already running and that the server can start running.
	WebServer.java	Listens on a specified server connection and waits for incoming messages. Listens for serial data coming from the AppletCenter. Once a message has arrived it takes the corresponding actions.

FIG. 35d

Functional Block	Class Name	Description
Replenisher	Client.java	This class has the ability to bind to ports on certain terminals and send out data for them. Sends out the Service Agent and the Alert Agents when desired.
	Server.java	Listens on a specified server connection and waits for incoming messages. Listens for Alert Agents coming to check out the Replenishers details. Once a message has arrived it takes the corresponding actions.
	RepDetails.java	Used to store the details a Replenisher has added to the details screen when they logged in.
	DetailsScreen.java	Main class for this package. Used to take the information from the screen and store it in RepDetails. It will also start-up the server and call the Client class to send out the Service Agents.

FIG. 35e

Functional Block	Class Name	Description
Agent	AgentHandler.java	Used to check that the thread is not already running and that the server can start running.
	AgentServer.java	Listens on a specified server connection and waits for incoming messages. Listens for incoming Service and Monitor Agents. Once a message has arrived it takes the corresponding actions.
	AgentClient.java	This class has the ability to bind to ports on certain terminals and send out data for them. Sends out Alert and Monitor Agents.

FIG. 35f

Functional Block	Class Name	Description
WebServer	Client.java	This class has the ability to bind to ports on certain terminals and send out data for them. Sends out all the data that is required to display information on the Applet.
	Server.java	Listens on a specified server connection and waits for incoming messages. Listens for requests from the Applet that are passed on by the Regional Server. Once a message has arrived it takes the corresponding actions.
	WebServer.java	Main class for this package, used to set-up and start the server.
	ErrorList.java	Class that holds the array of errors that have been produced and location details about where the errors occurred.
	ResetDetails.java	Class used to reset details in order to set a tally back to zero.

FIG. 35g

Functional Block	Class Name	Description
Tallies	Tallys.java	Inherited by all the classes below. Used in the Regional Server to access their functions and get the necessary information.
	CardTallys.java	Holds the tally mnemonics that are unique to the Card Reader class and the procedures that are needed to process them.
	DispenserTallys.java	Holds the tally mnemonics that are unique to the Cash Dispenser class and the procedures that are needed to process them.
	PrinterTallys.java	Holds the tally mnemonics that are unique to the Receipt Printer class and the procedures that are needed to process them.
	ProcessIPs.java	Used to get the modules Internet address and process it.

FIG. 36

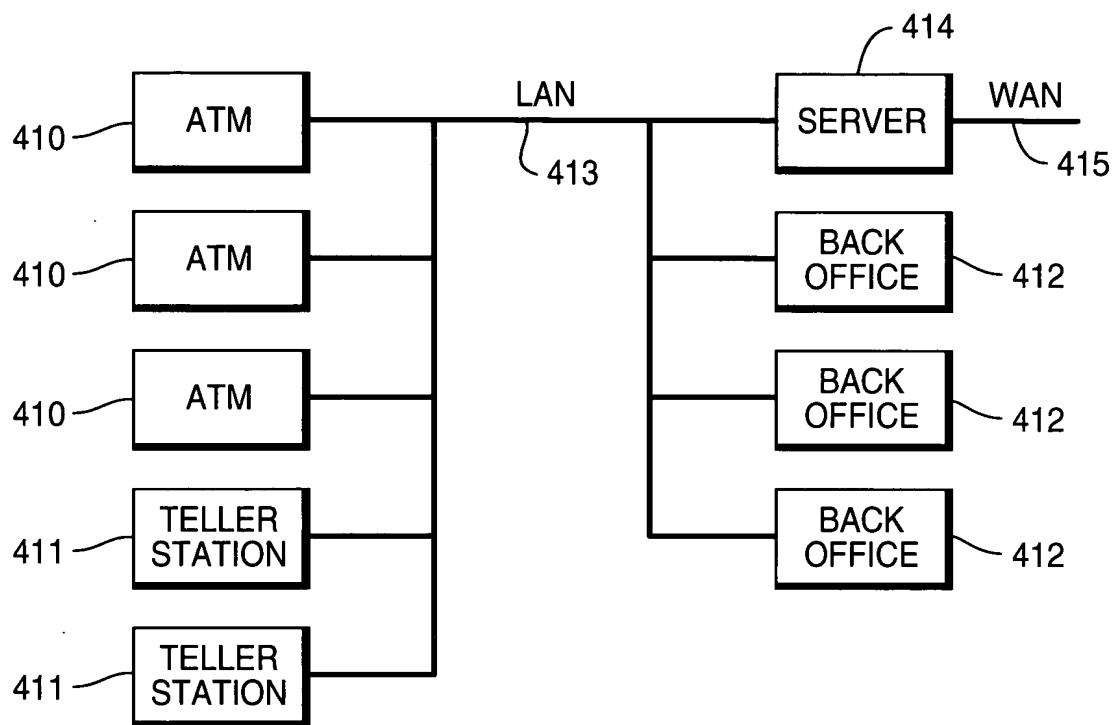


FIG. 38

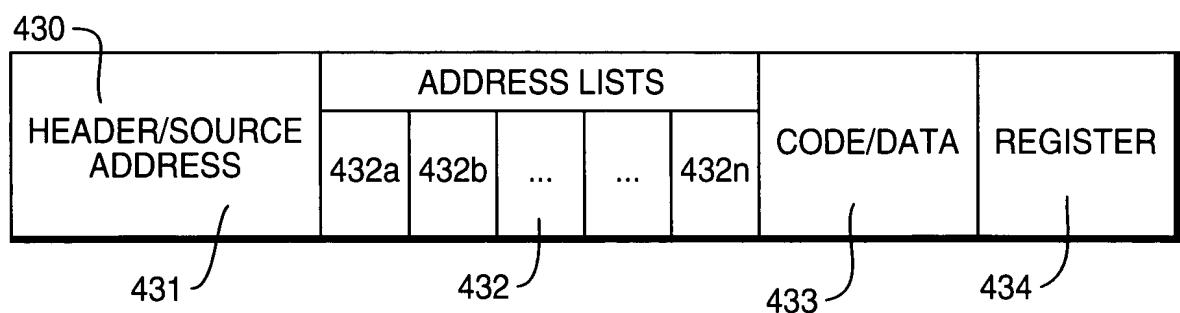


FIG. 37

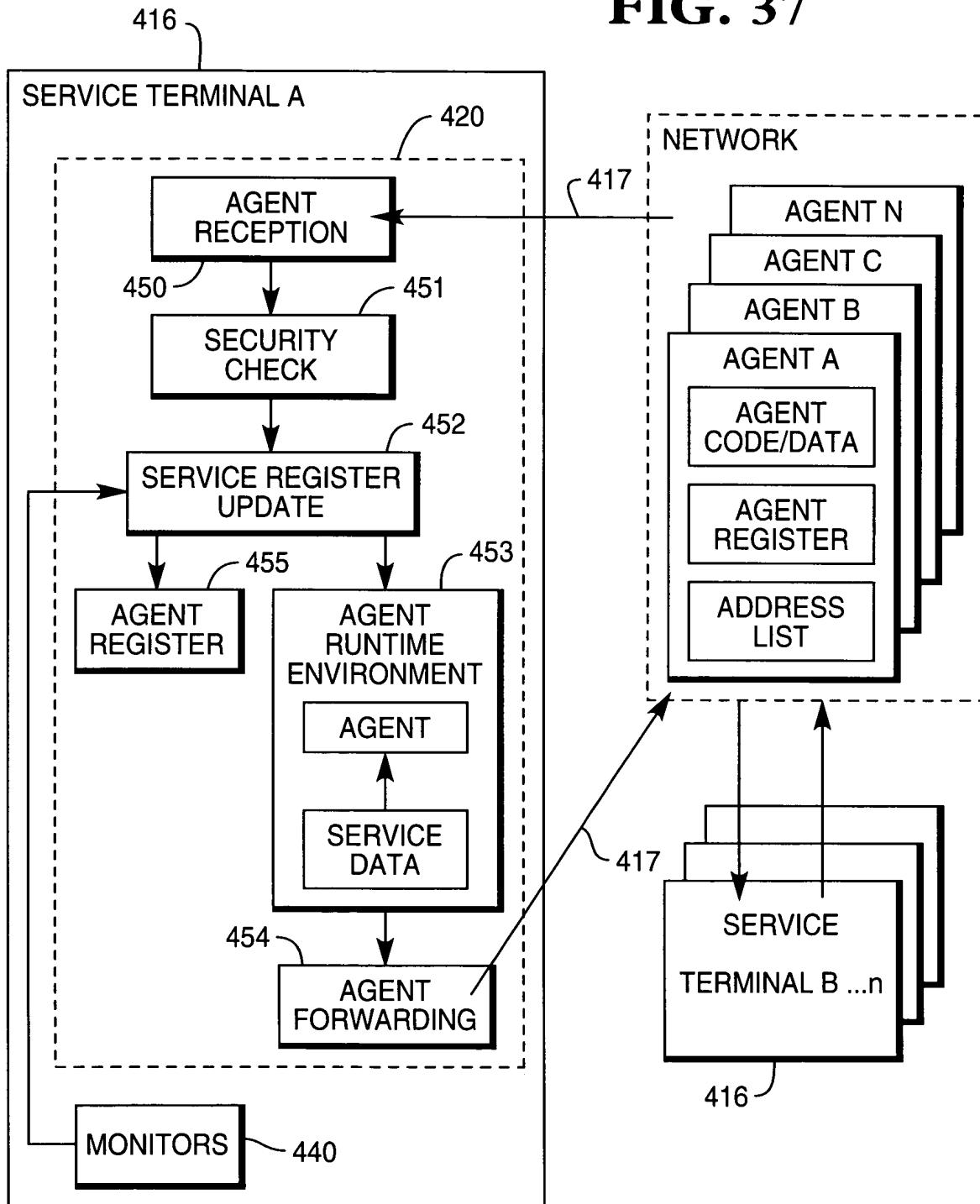


FIG. 39a

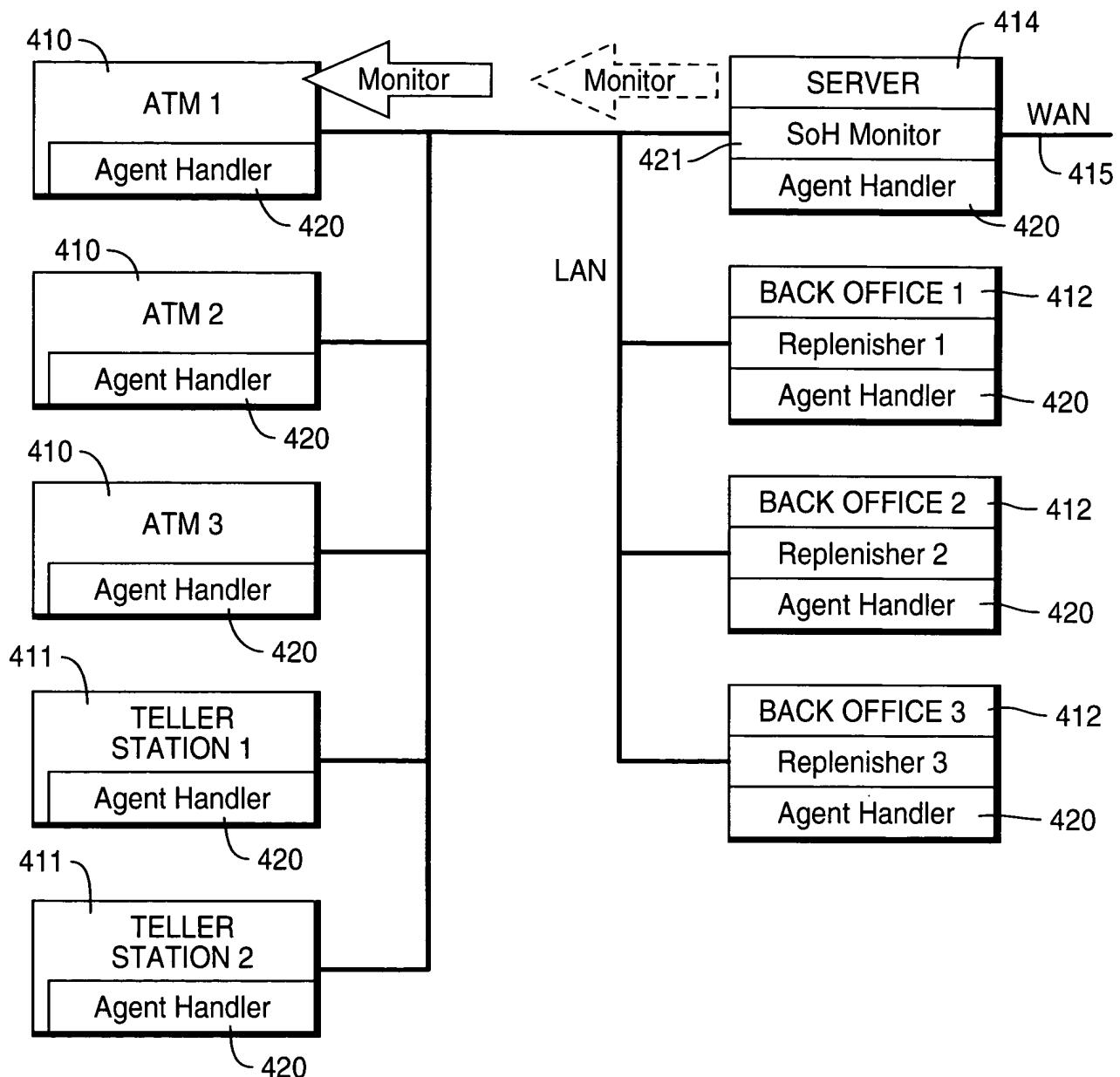


FIG. 39b

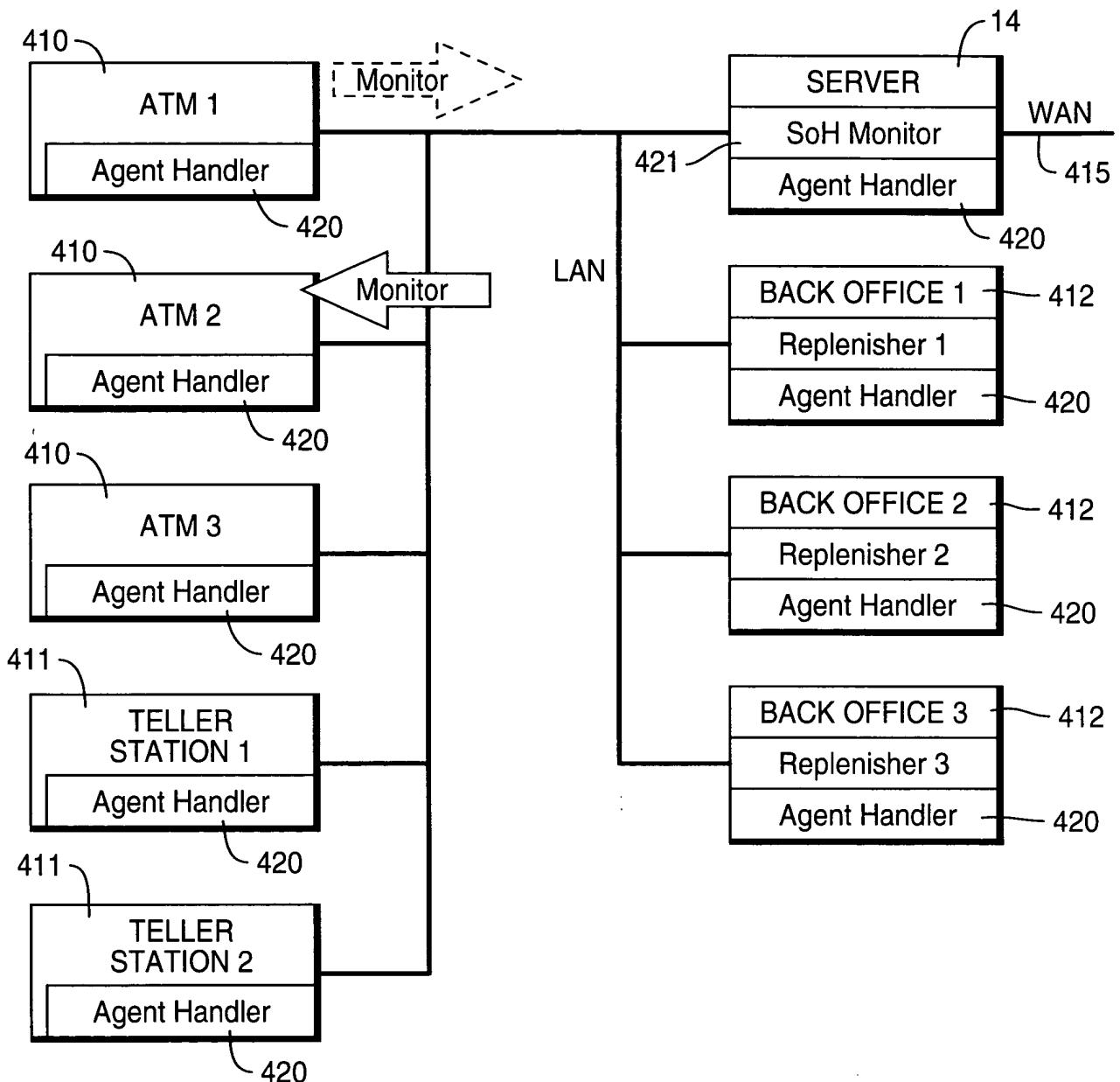


FIG. 39c

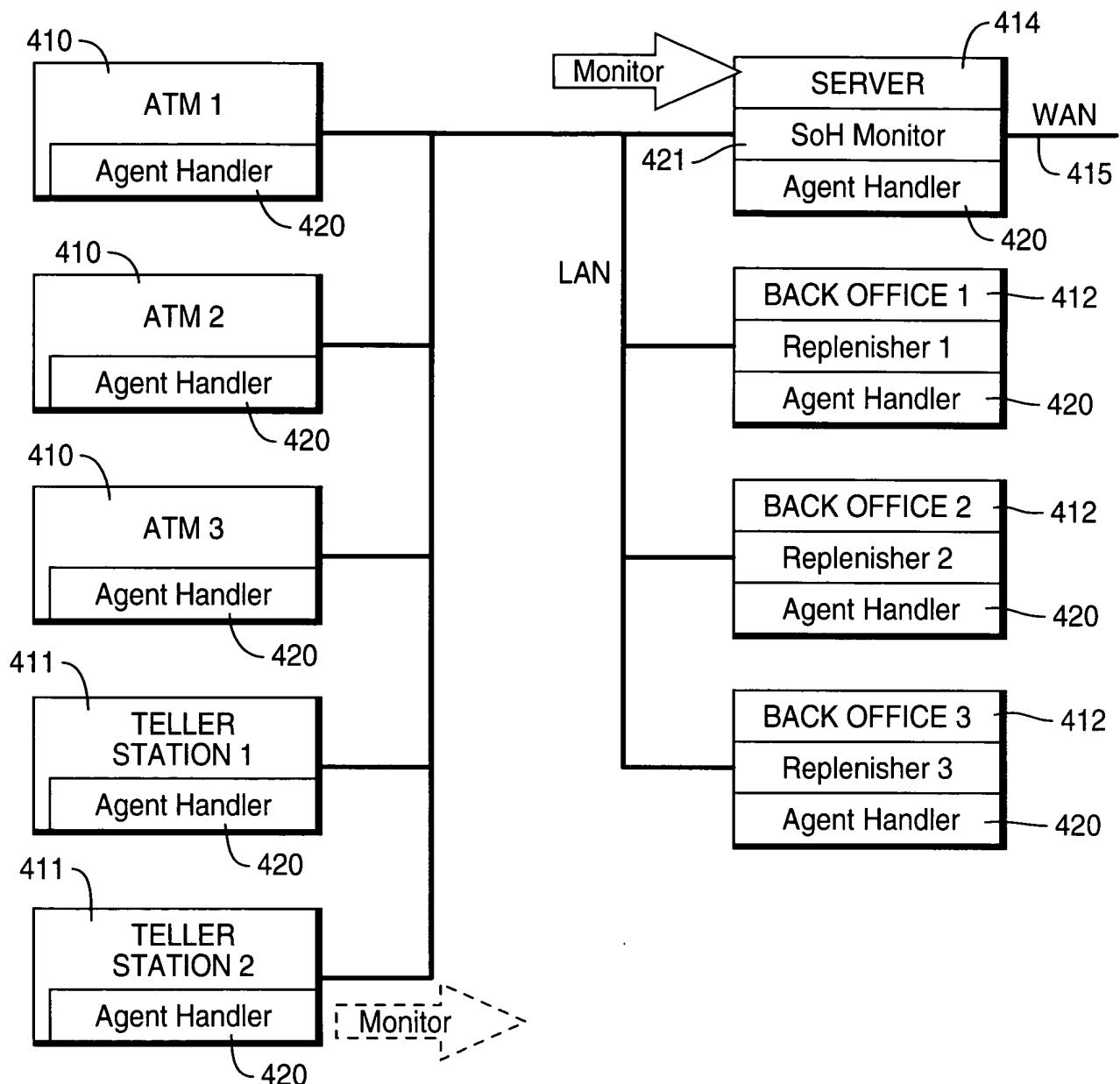


FIG. 40a

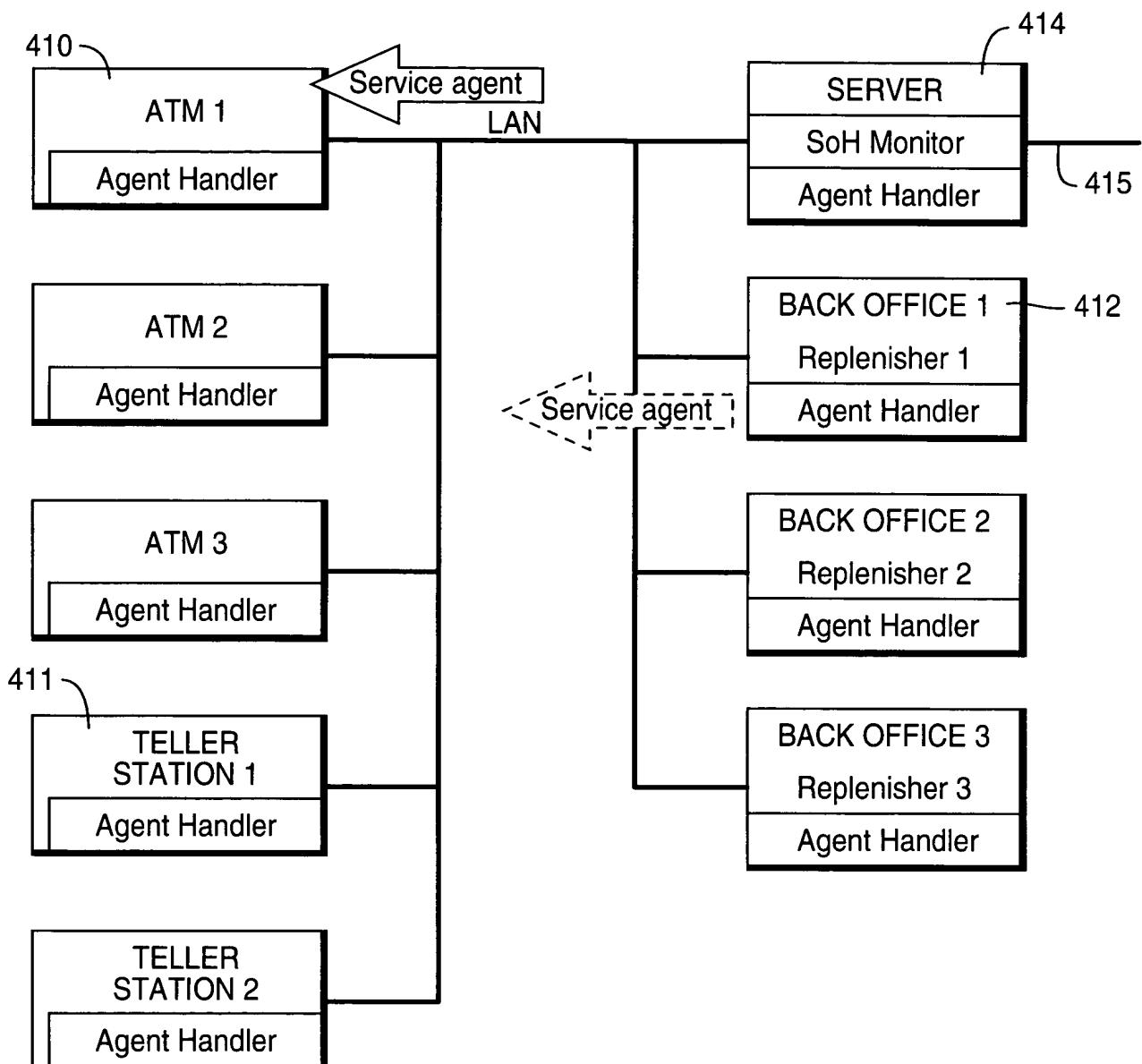


FIG. 40b

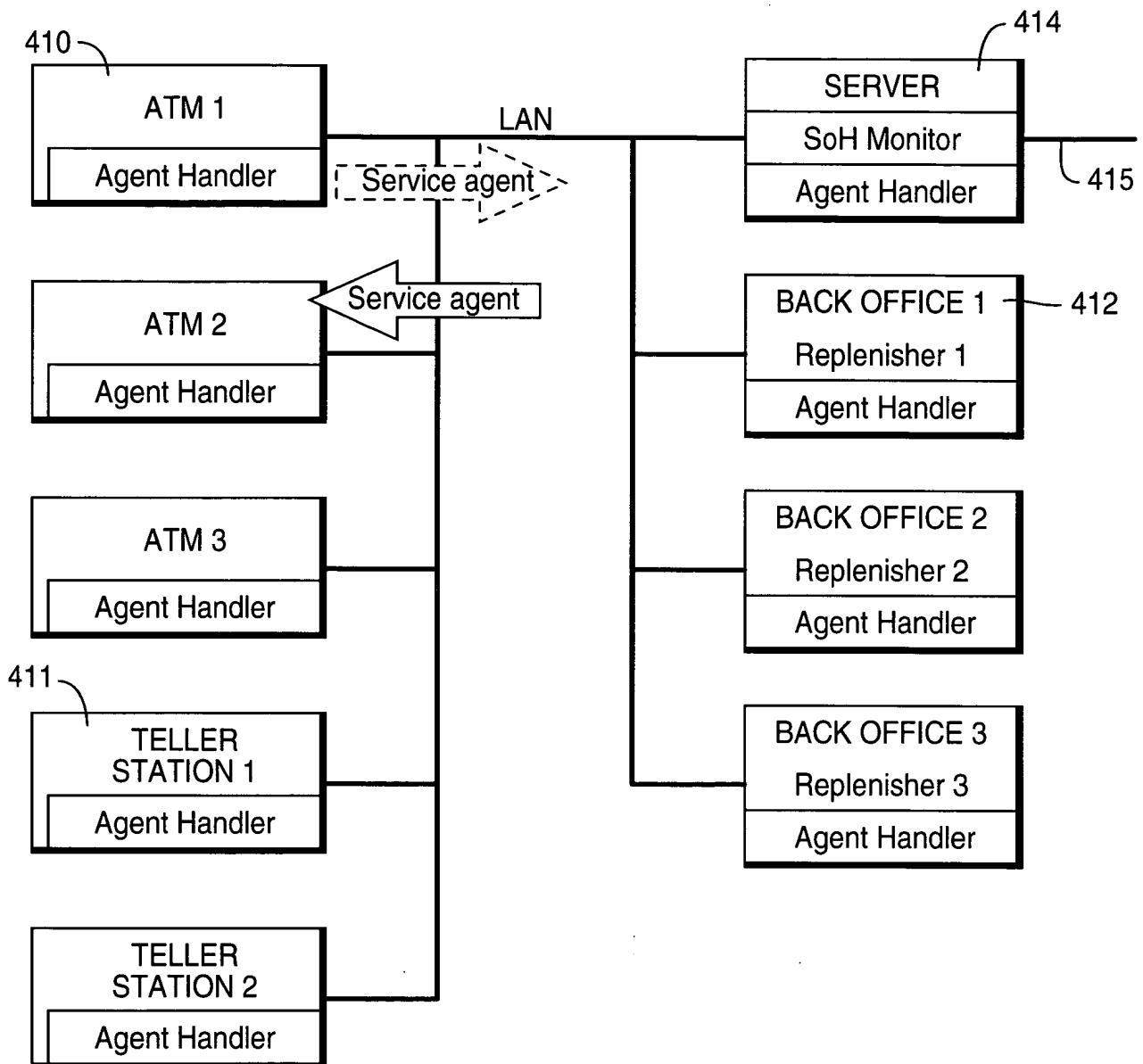


FIG. 40c

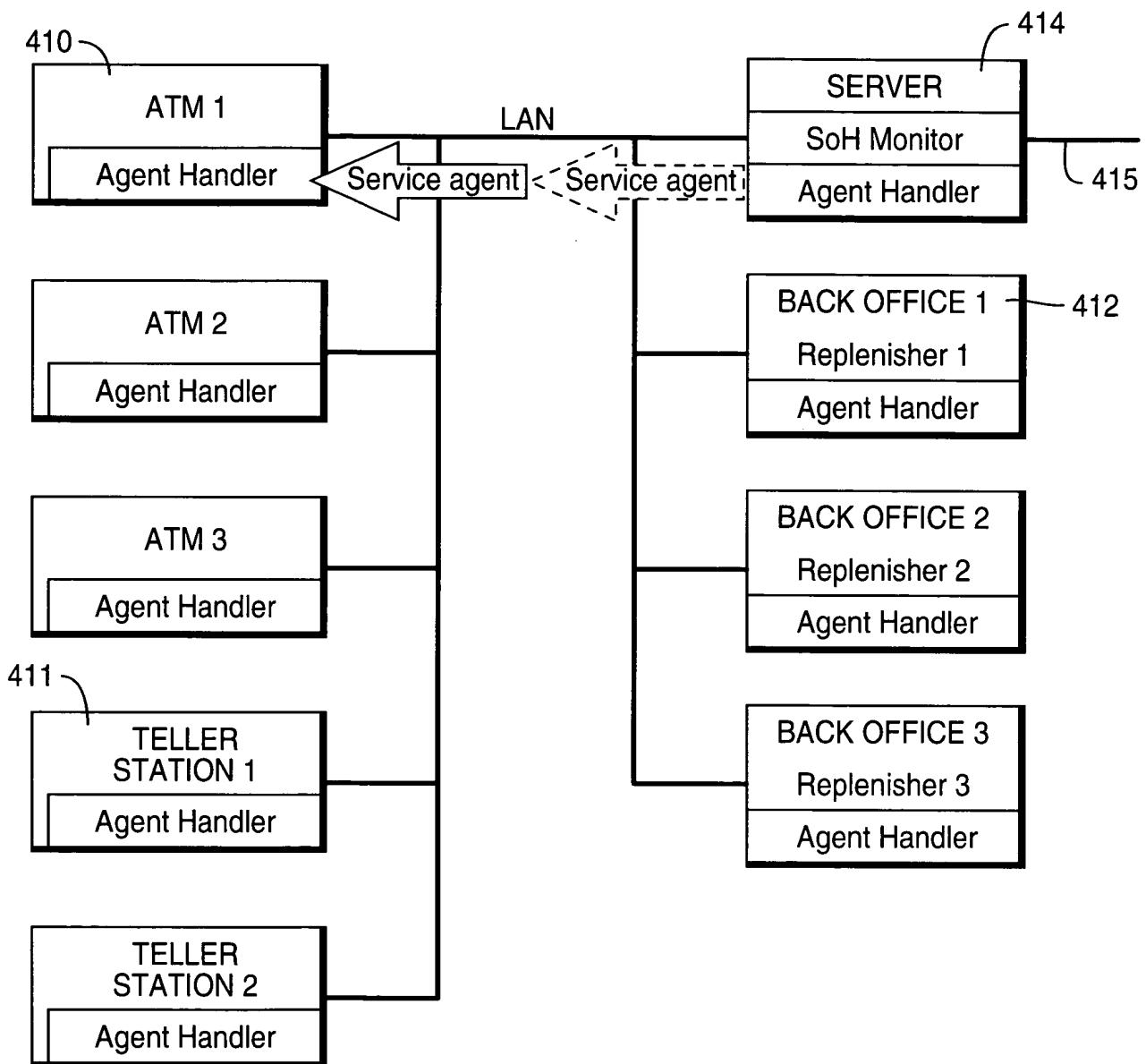


FIG. 41a

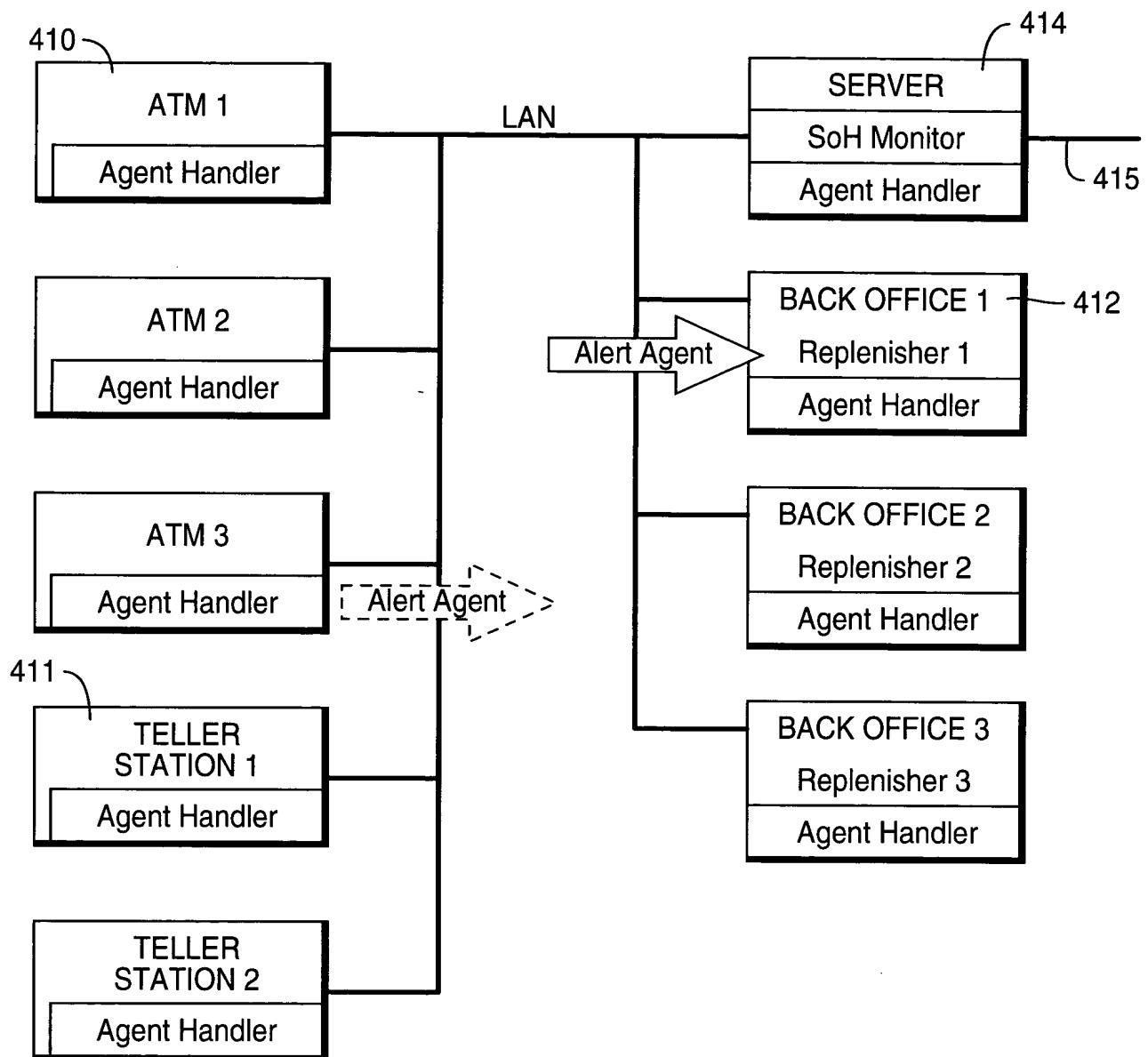


FIG. 41b

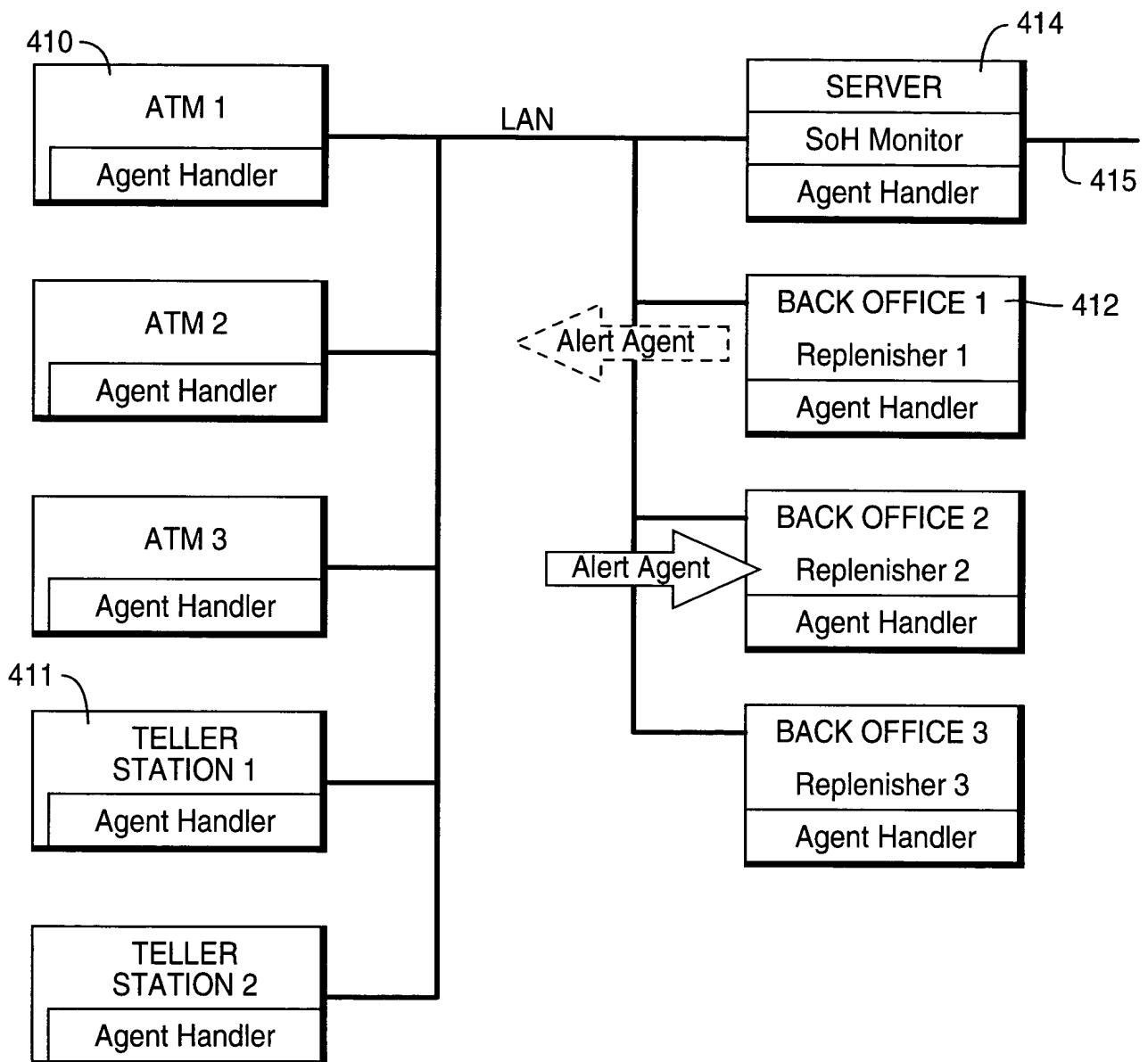


FIG. 41c

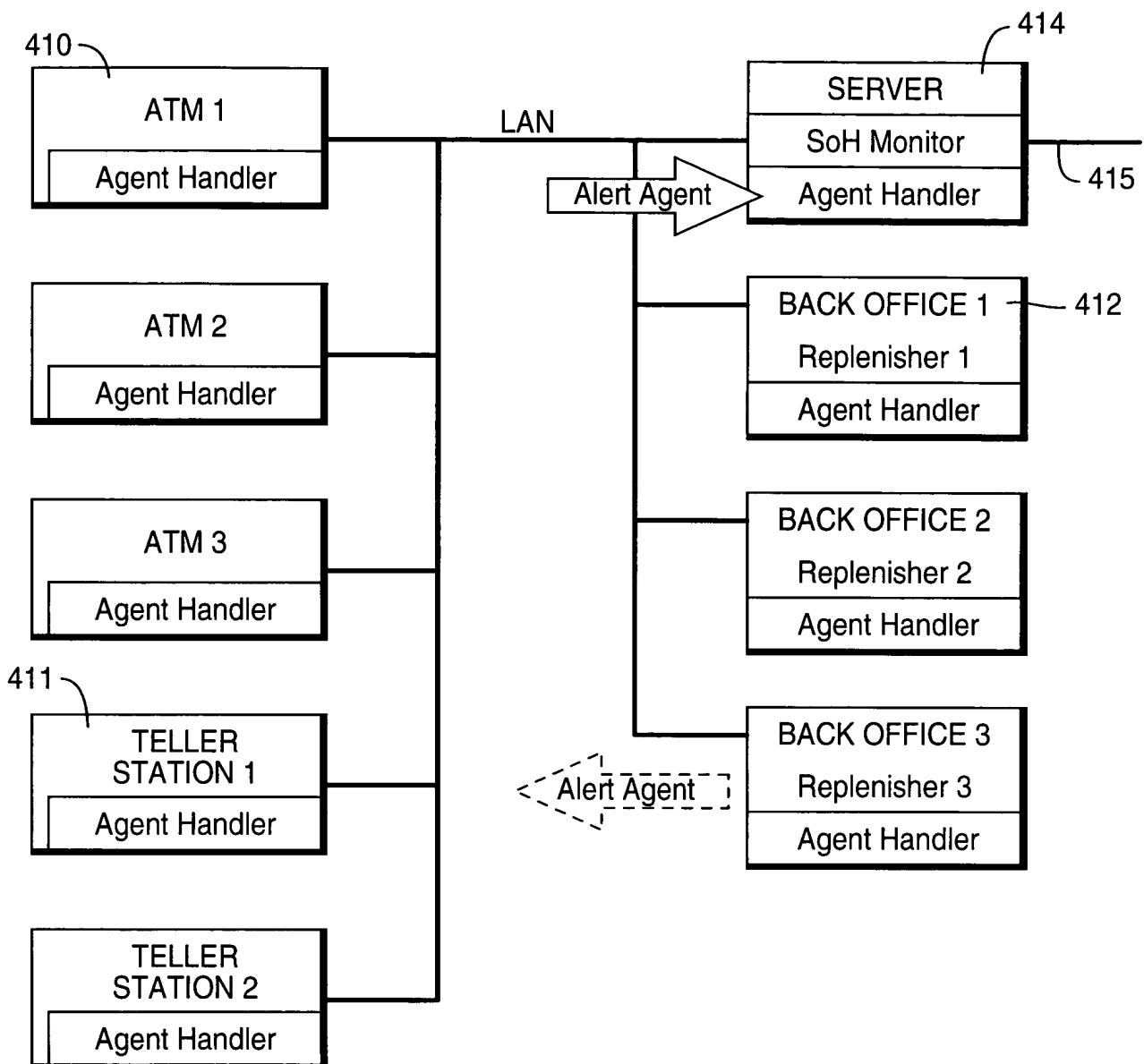


FIG. 42a

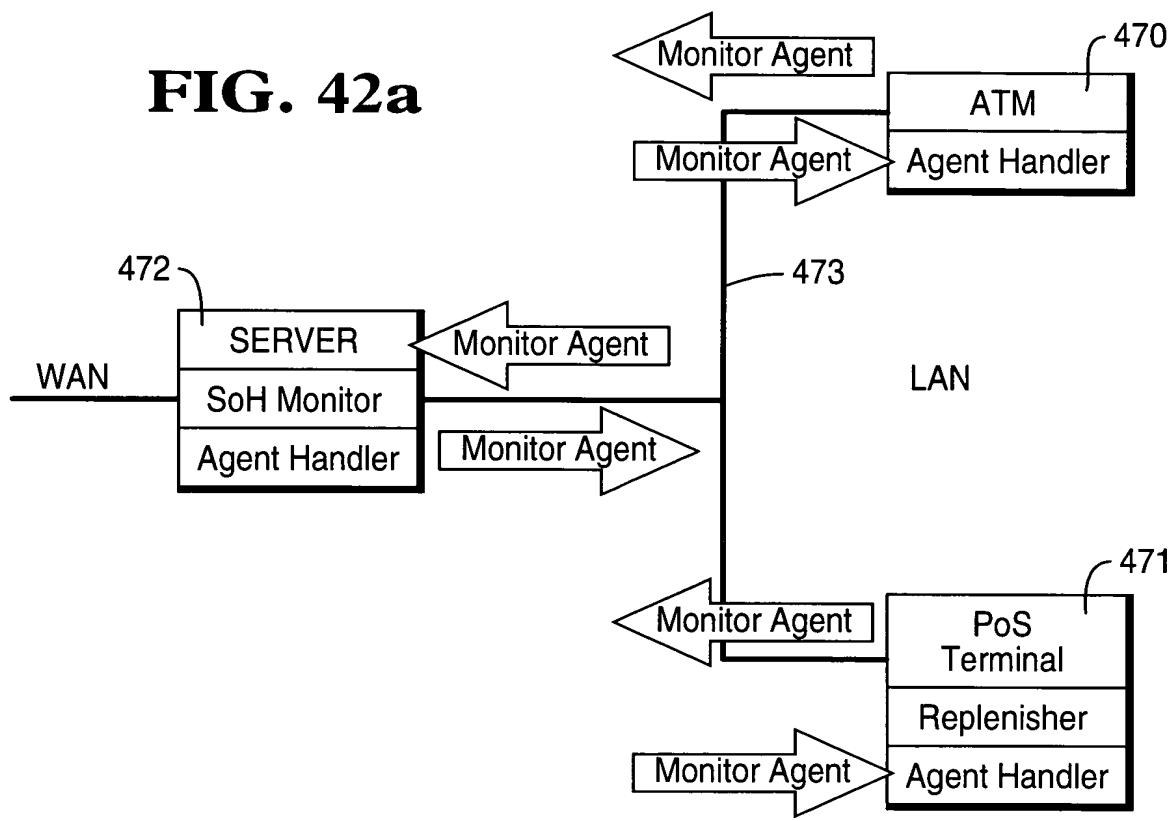


FIG. 42b

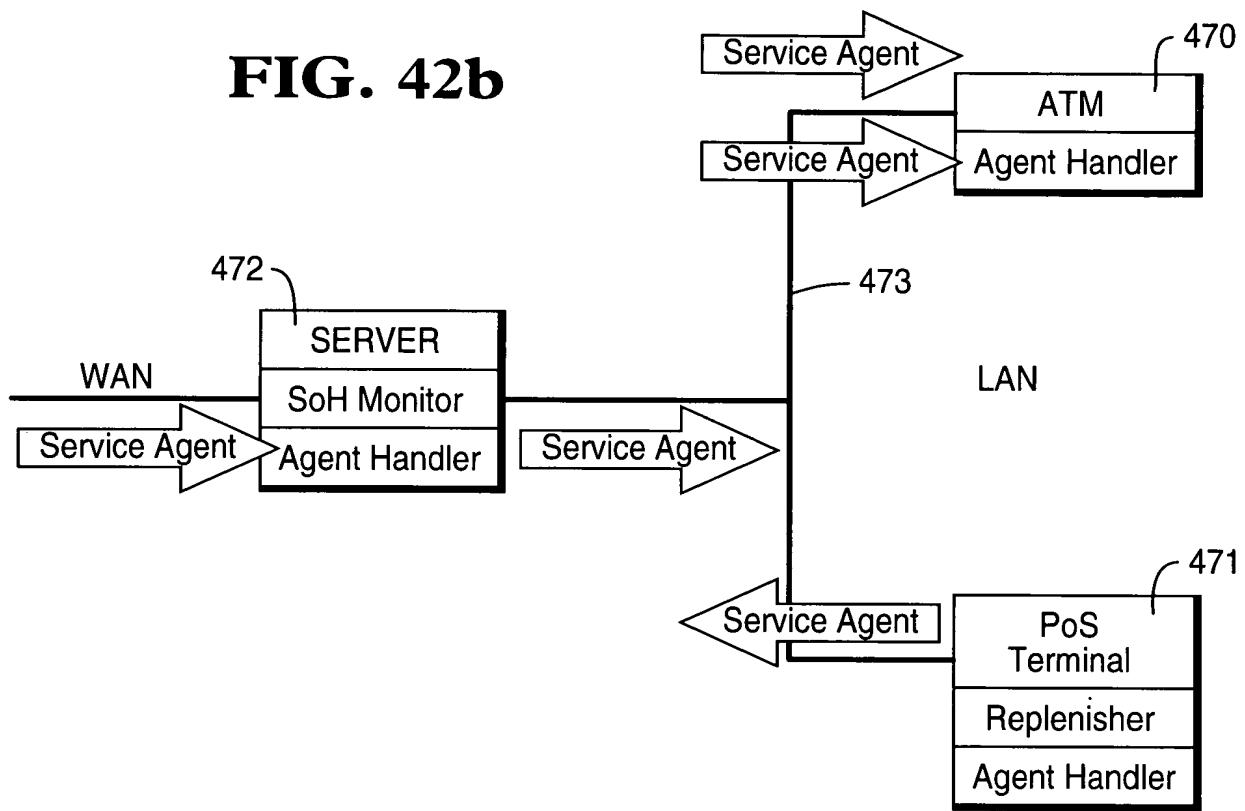


FIG. 42c

